Modélisation et manipulation d'entrepôts de données complexes et historisées

Olivier Teste

Laboratoire IRIT - Pôle SIG

THESE

présentée devant
l'UNIVERSITE PAUL SABATIER DE TOULOUSE (SCIENCES)

en vue de l'obtention du
DOCTORAT DE L'UNIVERSITE PAUL SABATIER
Spécialité : INFORMATIQUE

par

Olivier TESTE

Modélisation et manipulation d'entrepôts de données complexes et historisées

---

Soutenue le 18 décembre 2000 devant le jury composé de :

C. Cauvet	Professeur à l'Université Aix-Marseille III, rapporteur
C. Chrisment	Professeur à l'Université Toulouse III
A. Flory	Professeur à l'INSA de Lyon, rapporteur
J. Luguet	Professeur à l'Université Toulouse III
E. Métais	Professeur au CNAM de Paris, rapporteur
F. Ravat	Maître de Conférences à l'Université Toulouse I
G. Zurfluh	Professeur à l'Université Toulouse I, directeur de thèse

INSTITUT DE RECHERCHE EN INFORMATIQUE DE TOULOUSE
Centre National de la Recherche Scientifique (UMR 5505) - Institut National Polytechnique - Université Paul Sabatier
Université Paul Sabatier, 118 route de Narbonne, 31062 Toulouse cedex 04, Tel. 05.61.55.66.11

à mes parents

REMERCIEMENTS

Je tiens à exprimer toute ma reconnaissance à Messieurs Claude Chrisment, Jacques Luguet et Gilles Zurfluh, pour m'avoir accueilli au sein de leur équipe. Je les remercie et tiens à leur assurer ma profonde gratitude.

Je tiens à remercier très sincèrement l'ensemble des membres du jury qui me font le grand honneur d'avoir accepté de juger mon travail.

Je remercie Madame Corinne Cauvet, Professeur à l'Université d'Aix-Marseille III, pour l'honneur qu'elle me fait en acceptant d'être rapporteur de mon travail et pour ses remarques qui ont permis d'améliorer la qualité de ce mémoire. Je tiens à lui exprimer mes remerciements pour l'honneur qu'elle me fait en participant à ce jury.

Je remercie Madame Elisabeth Métais, Professeur au CNAM de Paris, pour l'honneur qu'elle me fait en acceptant d'être rapporteur de mes travaux et pour ses observations qui ont contribué à améliorer ce mémoire. Je tiens également à la remercier pour l'honneur qu'elle me fait en participant à ce jury.

Je remercie Monsieur André Flory, Professeur à l'INSA de Lyon, pour avoir accepté d'être rapporteur de mes recherches et pour ses remarques qui m'ont permis d'améliorer la qualité de ce mémoire. Je tiens à lui exprimer mes remerciements pour l'honneur qu'il me fait en participant à ce jury.

Je remercie Monsieur Gilles Zurfluh, Professeur à l'Université Toulouse I et directeur de ma thèse, pour toute la confiance qu'il m'a témoignée tout au long de ces années et sa constante disponibilité. Ses remarques constructives ont contribué à améliorer les travaux de recherche présentés dans ce mémoire. Qu'il soit ici assuré de ma profonde gratitude et de mon très grand respect.

Je remercie Monsieur Franck Ravat, Maître de Conférences à l'Université Toulouse I, pour son soutien et sa collaboration de tous les instants. Son aide et sa disponibilité ainsi que ses précieuses remarques ont grandement contribué à améliorer la qualité de ce mémoire. Qu'il trouve donc ici l'assurance de ma profonde gratitude. Je tiens à souligner également ses qualités humaines qui ont contribué à tisser des liens d'amitié entre nous.

Je remercie Monsieur Claude Chrisment, Professeur à l'Université Toulouse III, pour l'intérêt qu'il a porté à mes travaux en examinant ce mémoire, pour ses conseils ainsi que pour l'honneur et le plaisir qu'il me fait en participant à ce jury.

Je remercie Monsieur Jacques Luguet, Professeur à l'Université Toulouse III, pour avoir examiné mon travail ainsi que pour l'honneur et le plaisir qu'il me fait en participant à ce jury.

Je remercie l'ensemble des participants au projet REANIMATIC qui ont mis à ma disposition des moyens matériels. En particulier, je tiens à exprimer mes sincères remerciements au Professeur Sylviane Schwer pour toute l'aide qu'elle m'a apportée et dont les nombreuses discussions m'ont permis d'améliorer mes travaux de recherches.

Je remercie aussi les personnes du CTi-Sud qui m'ont accueilli dans leur société, qui ont mis à ma disposition des moyens matériels et qui m'ont permis d'avoir un champ d'application pour mes recherches.

Je voudrais remercier tous mes amis et collègues de l'équipe SIG du laboratoire IRIT pour leur aide, leur soutien et leur gentillesse. J'exprime en particulier ma gratitude à Chantal, Christine, Florence, Josiane, Eric, Gilles et Moh pour toute l'aide qu'ils m'ont accordé. Je remercie aussi Max, Fred, Xavier, Farshad et Philippe pour leur aide et leur amitié. Mes remerciements vont également à Faiza et à tous les autres membres de l'équipe.

Je remercie les membres du CERISS et de l'UT1 qui m'ont accueilli cette année.

Je voudrais également exprimer mes remerciements aux personnes extérieures au monde universitaire qui m'ont soutenu. En particulier, je remercie tous mes amis avec lesquels j'ai passé des moments inoubliables…

Je remercie tout particulièrement mes parents qui m'ont toujours soutenu et qui m'ont permis de mener à bien mes études. Je tiens à remercier également ma sœur Isabelle qui m'a supporté de nombreuses années. Enfin, je souhaite remercier celle qui me supporte encore.

SOMMAIRE

Table des matières

INTRODUCTION

Contexte général de l'étude

Face à la mondialisation et à la concurrence grandissante, la prise de décision est devenue cruciale pour les dirigeants d'entreprises. L'efficacité de cette prise de décision repose sur la mise à disposition d'informations pertinentes et d'outils adaptés. Le problème des entreprises est d'exploiter efficacement d'importants volumes d'informations, provenant soit de leurs systèmes opérationnels, soit de leur environnement extérieur, pour **supporter la prise de décision**.

Les systèmes traditionnels s'avèrent inadaptés à une telle activité [Codd 1993] [Inmon 1994] [Kimball 1996]. Afin de pallier cet inconvénient, des **systèmes décisionnels** ont été développés ; selon le "Meta Group", 95% des 500 entreprises les plus importantes aux Etats-Unis ont déjà ou sont en train de finaliser la mise en place d'un tel système. La plupart de ces systèmes reposent sur un espace de stockage centralisé, appelé **entrepôt de données** (*data warehouse*) ; son rôle est d'intégrer et de stocker l'information utile aux décideurs et de conserver l'historique des données pour supporter les analyses effectuées lors des prises de décision [Inmon 1994] [Widom 1995] [Chaudhuri, Dayal 1997].

Les offres commerciales actuelles proposent des logiciels d'extraction et d'analyse ; ils permettent de collecter des informations provenant de sources différentes et d'exploiter ces données au travers d'interfaces utilisateurs. Néanmoins ces outils ne proposent pas une solution intégrée de conception et de développement des systèmes décisionnels. De plus, ils ne répondent pas entièrement aux caractéristiques des applications actuelles telles que la modélisation des données complexes et la gestion de leur évolution [Gatziu, et al 1999] [Pedersen, Jensen 1999].

Problématique et cadre applicatif

Notre objectif est d'apporter des solutions nouvelles pour la modélisation et le développement d'entrepôts. Face à la profusion d'informations hétérogènes, la conception et le développement de systèmes décisionnels adaptés s'avèrent primordiaux [Pedersen, Jensen 1998]. Le cadre applicatif de nos travaux se situe dans le domaine médical.

Par nature, les applications médicales imposent des structures de **données complexes** [Pedersen, Jensen 1998]. Par exemple, le dossier patient regroupe un ensemble d'enregistrements variés et hétérogènes : il comporte des données administratives, des données sur les traitements médicaux, des données relatives aux diagnostics et aux analyses effectuées… La complexité de ces structures est difficile à prendre en compte avec les modèles actuels pour des raisons essentiellement dues à leur faible niveau d'abstraction [Pedersen, Jensen 1998, 1999].

Par ailleurs, les applications décisionnelles médicales (et plus généralement toutes les applications décisionnelles) utilisent fréquemment des **données temporelles** [Chaudhuri, Dayal 1997] [Pedersen, Jensen 1998]. Malgré l'intérêt que portent les décideurs aux

évolutions des données [Chaudhuri, Dayal 1997], les systèmes commerciaux actuels n'intègrent pas l'historisation des données dans les entrepôts. En outre, peu de travaux de recherche sur les entrepôts traitent de cet aspect [Pedersen, Jensen 1999] [Yang, Widom 1998, 2000] [Mendelzon, Vaisman 2000].

Enfin, l'importance des volumes de données mis en jeu dans les systèmes d'aide à la décision nécessite des mécanismes d'agrégation pour synthétiser l'information. Ces mécanismes sont d'autant plus importants qu'il est crucial d'offrir des moyens pour **archiver** les données temporelles à des niveaux de détail en adéquation avec les besoins des décideurs [Inmon 1994].

Nos travaux de recherche s'inscrivent dans le cadre du laboratoire **IRIT** (Institut de Recherche en Informatique de Toulouse) au sein de l'équipe **SIG** (Systèmes d'Informations Généralisées). Notre équipe ayant acquis une expérience dans le domaine de la conception et du développement de systèmes d'informations complexes, évolutifs et répartis, est co-initiatrice du projet **EVOLUTION**. Ce projet, soutenu financièrement par le Ministère de l'Education Nationale, de la Recherche et de la Technologie (MENRT), fédère les équipes françaises de recherche et vise au développement d'une méthodologie et d'outils de type CASE pour l'aide à la conception et l'évolution des entrepôts de données.

Cette expérience nous a permis de répondre aux besoins exprimés au cours de différentes collaborations dans le milieu médical.

- La première collaboration s'est déroulée dans le cadre d'une convention entre l'Université Paul Sabatier et la société **CTI-Sud** [Ctisud 1998] (Centre de traitement informatique de l'Assurance Maladie de Midi-Pyrénées et Languedoc-Roussillon), où nous avons mené une expertise sur l'élaboration d'un entrepôt de données médicales évolutives. L'objectif de l'entrepôt est d'améliorer le suivi et le contrôle des dépenses engagées par l'activité des praticiens de santé.
- La seconde collaboration concerne le projet **REANIMATIC** [Reanimatic 1999] qui met en commun les compétences de médecins réanimateurs et de chercheurs, afin de concevoir un système décisionnel évolutif de veille des patients en réanimation. L'objet de ce système est de supporter des estimations sur la probabilité de survie et sur les probabilités de survenues de complications, afin d'améliorer la qualité des soins et le devenir des patients en réanimation.

Contributions de nos travaux de recherche

Notre approche se base sur une **dichotomie** entre l'**entrepôt de données** qui centralise toute l'information pertinente pour la prise de décision et des **magasins de données** qui constituent un extrait de l'entrepôt pour répondre à un besoin d'analyse particulier. **L'objet de nos travaux est donc de spécifier des modèles de représentation et des langages de manipulation dédiés aux entrepôts et aux magasins de données complexes et évolutives.** Peu de recherches traitent de cette double problématique dans le cadre des systèmes décisionnels : fournir des modèles à données complexes et évolutives [Pedersen, Jensen 1998, 1999] et spécifier les langages de manipulation associés [Mendelzon, Vaisman 2000].

Le modèle de données pour les entrepôts que nous définissons, à la différence de ceux proposés dans la littérature, est un modèle conceptuel. Il permet de décrire au sein de l'entrepôt des entités extraites d'une source pour lesquelles l'évolution est conservée. Ce modèle intègre un processus d'archivage automatique des données temporelles permettant de réduire le volume des données tout en conservant les données utiles aux décideurs. L'élaboration de l'entrepôt est réalisée au travers d'un processus semi-automatique d'extractions appliquées sur une source de données. L'originalité de notre approche vient du traitement de l'aspect statique des données et de l'aspect dynamique qui n'est habituellement pas traité dans les autres travaux. L'algèbre proposée pour l'interrogation de l'entrepôt se compose d'un ensemble d'opérateurs qui permet d'interroger de façon uniforme les objets et leurs états passés ou archivés. Les opérations algébriques proposées regroupent des opérateurs issus des algèbres objet et des algèbres temporelles ainsi que des opérateurs spécifiques à notre modélisation. D'autres opérateurs permettent de transformer les données de l'entrepôt en séries temporelles pour offrir différents points de vues sur les données de manière à faciliter leur analyse.

Enfin, nous définissons un modèle conceptuel de représentation pour les magasins de données multidimensionnelles ; ce modèle est associé à une algèbre pour l'interrogation des magasins. Notre modélisation constitue une généralisation des modèles proposés dans la littérature tandis que notre algèbre intègre les principales opérations multidimensionnelles actuelles. Enfin, nous spécifions une démarche pour la transformation des données de l'entrepôt vers une organisation multidimensionnelle dans les magasins de données. Cette démarche constitue une première contribution au développement d'une méthode de conception de schémas multidimensionnels complexes.

Organisation du mémoire

Pour présenter nos travaux et le domaine dans lequel ils s'inscrivent, nous avons retenu pour ce mémoire de thèse une organisation en six chapitres.

Dans le premier chapitre, nous précisons le cadre de cette thèse en définissant les concepts de base des systèmes d'aide à la décision. Nous présentons les principes de la modélisation multidimensionnelle et nous distinguons les concepts d'entrepôt et de magasin de données. Nous effectuons un état de l'art concernant la recherche actuelle dans le domaine des entrepôts ; nous montrons que la modélisation et l'évolution des données stockées dans les entrepôts sont peu étudiées. Nous présentons également les principales solutions proposées actuellement dans les bases de données pour conserver l'évolution des données. Enfin, nous positionnons nos travaux dans notre architecture pour les systèmes décisionnels et nous définissons notre problématique de recherche.

Dans le deuxième chapitre nous étudions la modélisation conceptuelle des entrepôts de données. Notre modèle permet la représentation d'objets au travers d'états courants, passés et archivés, permettant de conserver les données actuelles mais également leurs évolutions sous forme détaillée ou résumée. En outre, ce modèle unifie les niveaux d'historisation en définissant le concept d'environnement.

Dans le troisième chapitre, nous traitons de l'élaboration des entrepôts par extraction. Cette élaboration est effectuée tant sur le plan statique que sur le plan dynamique des données.

Dans le quatrième chapitre, nous présentons la manipulation des données stockées dans l'entrepôt. Nous définissons une algèbre qui tout en reprenant les principaux opérateurs des algèbres objet et temporelles, propose de nouveaux opérateurs spécifiques aux objets de l'entrepôt.

Dans le cinquième chapitre, nous étudions la modélisation et l'élaboration des magasins de données. Nous proposons un modèle conceptuel multidimensionnel qui généralise les différents modèles multidimensionnels. Enfin, nous spécifions une démarche de conception permettant l'élaboration des magasins multidimensionnels à partir d'un entrepôt de données.

Dans le sixième chapitre, nous décrivons les implantations réalisées pour valider nos propositions. Nous présentons un outil d'aide à la définition et à la génération d'entrepôts de données ; celui-ci repose sur les concepts que nous avons défini pour la modélisation et l'élaboration des entrepôts. Il permet de produire graphiquement et de manière interactive un entrepôt à partir d'un graphe représentant une source de données.

CHAPITRE I :
ETAT DE L'ART ET PROPOSITIONS

1 INTRODUCTION

Avec la généralisation de l’informatique dans tous les secteurs d’activité, les entreprises produisent et manipulent de très importants volumes de données électroniques. Ces données sont stockées dans les systèmes opérationnels de l'entreprise au sein de bases de données, de fichiers… L’exploitation de ces données dans un but d'analyse et de support à la prise de décision s’avère difficile et fastidieuse ; elle est réalisée le plus souvent de manière imparfaite par les décideurs grâce à des moyens classiques (requêtes SQL, vues, outils graphiques d'interrogation…).

Ces systèmes paraissent peu adaptés pour servir de support à la prise de décision. Ces bases opérationnelles utilisent le modèle relationnel ; celui-ci convient bien aux applications gérant l'activité quotidienne de l'entreprise, mais s'avère inadapté au décisionnel [Codd 1993] [Kimball 1996]. Face à cette inadéquation, les entreprises ont recours à des systèmes d'aide à la décision spécifiques, basés sur l'approche des entrepôts de données. Cependant, de tels systèmes restent difficiles à élaborer et sont souvent réalisés de manière empirique, rendant l'évolution du système décisionnel délicate. Actuellement, les entreprises ont besoin d'outils et de modèles pour la mise en place de systèmes décisionnels comportant des données évolutives [Inmon 1994] [Pedersen, Jensen 1999] [Mendelzon, Vaisman 2000].

L'objet de ce chapitre est de définir les concepts inhérents aux systèmes décisionnels et de présenter la problématique de nos recherches.

- La section 2 définit les concepts fondamentaux relatifs aux systèmes décisionnels.
- La section 3 présente les axes de recherche principaux dans le domaine des entrepôts de données ainsi qu'un état de l'art concernant la technique des vues matérialisées et de la modélisation multidimensionnelle.
- La section 4 décrit les approches existantes dans les systèmes de gestion de bases de données pour conserver les évolutions des données ; également les bases de données temporelles et les bases de données gérant des versions sont abordées.
- Dans la section 5, nous identifions les limites des travaux actuels et nous présentons notre proposition.

2 LES CONCEPTS DE BASE

Après avoir présenté les composants d'un système d'aide à la décision, nous étudions les concepts d'entrepôt et de magasin de données.

2.1 Systèmes OLTP versus systèmes OLAP

Les bases de données sont utilisées dans les entreprises pour gérer les importants volumes d'informations contenus dans leurs systèmes opérationnels. Ces données sont gérées selon des

processus transactionnels en ligne (OLTP : "*On-Line Transactional Processing"* [Codd 1993]) qui se caractérisent de la manière suivante [Codd 1993] [Inmon 1994] [Kimball 1996] [Chaudhuri, Dayal 1997] :

- ils sont nombreux au sein d'une entreprise,
- ils concernent essentiellement la mise à jour des données,
- ils traitent un nombre d'enregistrements réduit,
- ils sont définis et exécutés par de nombreux utilisateurs.

L'exploitation de l'information contenue dans ces systèmes opérationnels est devenue une préoccupation essentielle pour les dirigeants des entreprises qui désirent améliorer leur prise de décision par une meilleure connaissance de leur propre activité, de celle de la concurrence, des employés, des clients et des fournisseurs. Les entreprises sont donc à la recherche de systèmes supportant efficacement les applications d'aide à la décision. Ces applications décisionnelles utilisent des processus d'analyse en ligne de données (OLAP : "*On-Line Analytical Processing"* [Codd 1993]). Ces processus répondent aux besoins spécifiques des analyses d'information. Dans [Codd 1993] E.F. Codd définit un cahier des charges comprenant douze règles que doivent satisfaire les systèmes décisionnels : l'analyse des données doit se faire de manière interactive et rapide, pour des données quelconques et historisées. Ces processus OLAP se caractérisent de la manière suivante [Codd 1993] [Inmon 1994] [Kimball 1996] [Chaudhuri, Dayal 1997] :

- ils sont peu nombreux, mais leurs données et traitements sont complexes,
- il s'agit uniquement de traitements semi-automatiques visant à interroger, visualiser et synthétiser les données,
- ils concernent un nombre d'enregistrements importants aux structures hétérogènes,
- ils sont définis et mis en œuvre par un nombre réduit d'utilisateurs qui sont les décideurs.

Le Tableau 1 compare les caractéristiques des processus OLTP et OLAP.

	Processus OLTP	**Processus OLAP**
Données	Exhaustives Courantes Dynamiques Orientées applications	Résumées Historiques Statiques Orientées sujets (d'analyse)
Utilisateurs	Nombreux Variés (employés, directeurs,…) Concurrents Mises à jours et interrogations Requêtes prédéfinies Réponses immédiates Accès à peu d'information	Peu nombreux Uniquement les décideurs Non concurrents Interrogations Requêtes imprévisibles et complexes Réponses moins rapides Accès à de nombreuses informations

Tableau 1 : Comparaison des processus OLTP et OLAP.

2.2 Systèmes décisionnels

Cette nouvelle utilisation de l'information contenue dans les bases opérationnelles des entreprises a donné lieu à l'élaboration de nouveaux systèmes dédiés à l'analyse et à la prise de décision. Ces systèmes sont désignés par le terme de systèmes décisionnels. Ils regroupent un

ensemble d'informations et d'outils mis à la disposition des décideurs pour supporter de manière efficace la prise de décision [Codd 1993] [Inmon 1994] [Chaudhuri, Dayal 1997].

Définition :
Un **système décisionnel** est un système d'information dédié aux applications décisionnelles.

L'architecture des systèmes décisionnels met en jeu quatre éléments essentiels : les sources de données, l'entrepôt de données, les magasins de données et les outils d'analyse et d'interrogation.

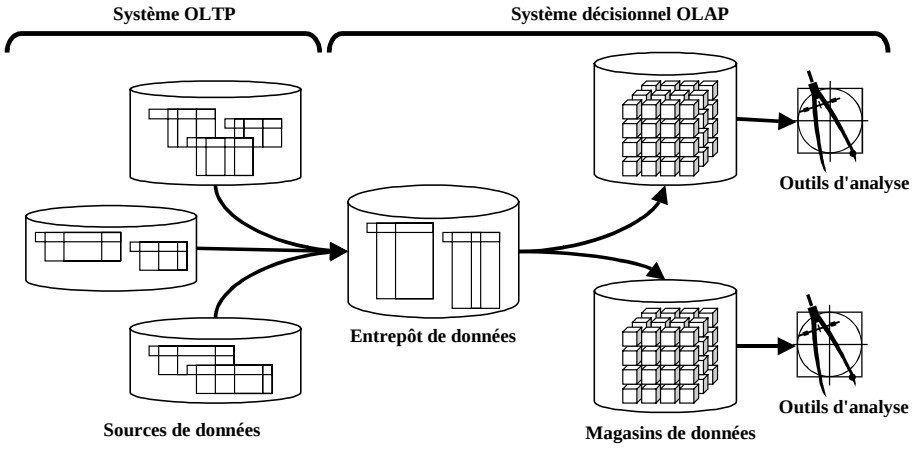


Figure 1 : Architecture des systèmes décisionnels.

Les **sources de données** sont nombreuses, variées, distribuées et autonomes. Elles peuvent être internes (bases de production) ou externes (Internet, bases des partenaires) à l'entreprise.

L'**entrepôt de données** est le lieu de stockage centralisé des informations utiles pour les décideurs. Il met en commun les données provenant des différentes sources et conserve leurs évolutions.

Les **magasins de données** sont des extraits de l'entrepôt orientés sujet. Les données sont organisées de manière adéquate pour permettre des analyses rapides à des fins de prise de décision.

Les **outils d'analyse** permettent de manipuler les données suivant des axes d'analyses. L'information est visualisée au travers d'interfaces interactives et fonctionnelles dédiées à des décideurs souvent non informaticiens (directeurs, chefs de services,…).

2.3 Entrepôts et magasins de données

Dans son ouvrage de référence "*Building the Data Warehouse*" [Inmon 1994], W.H. Inmon définit l'entrepôt de données comme "*une collection de données intégrées, orientées sujet, non volatiles, historisées, résumées et disponibles pour l'interrogation et l'analyse*". Cette définition englobe différents termes que nous explicitons.

- *« intégrées »*. Les données de l'entrepôt proviennent de différentes sources éventuellement hétérogènes. L'intégration consiste à résoudre les problèmes d'hétérogénéité des systèmes de stockage, des modèles de données, de sémantique de données.

- *« orientées sujet »*. Les données de l'entrepôt peuvent être réorganisées autour de thèmes tels que le patient, les diagnostics, les médicaments…
- *« non volatiles »*. Les données de l'entrepôt sont essentiellement utilisées en mode de consultation. Elles ne sont pas modifiées par les utilisateurs.
- *« historisées »*. La prise en compte de l'évolution des données est essentielle pour la prise de décision qui par exemple utilise des techniques de prédiction en s'appuyant sur les évolutions passées pour prévoir les évolutions futures.
- *« résumées »*. Les informations issues des sources de données doivent être agrégées et réorganisées afin de faciliter le processus de prise de décision.
- *« disponibles pour l'interrogation et l'analyse »*. Les utilisateurs doivent pouvoir consulter les données réorganisées de l'entrepôt en fonction de leur droit d'accès au travers d'outils interactifs d'aide à la manipulation et l'analyse .

Contrairement à l'architecture présentée à la Figure 1, cette définition correspond à un système décisionnel où l'entrepôt de données ne se distingue pas des magasins de données. Or, dans un système décisionnel tel que nous le définissons, l'objectif de ces deux espaces de stockage est différent et les problèmes à résoudre divergent : l'entrepôt de données regroupe toute l'information décisionnelle tandis que les magasins de données contiennent une partie de cette information, dédiée à un thème, un métier, une analyse…

Définitions :
L'**entrepôt de données** est le lieu de stockage centralisé d'un extrait des bases de production. Cet extrait concerne les données pertinentes pour le support à la décision. Elles sont intégrées et historisées. L'organisation des données est faite selon un modèle qui facilite la gestion efficace des données et leur historisation.

Le **magasin de données** est un extrait de l'entrepôt. Les données extraites sont adaptées à une classe de décideurs ou à un usage particulier (recherche de corrélation, logiciel de statistiques,...). L'organisation des données suit un modèle spécifique qui facilite les traitements décisionnels.

2.4 Modélisation multidimensionnelle

Suivant cette définition, les données à analyser au niveau d'un magasin doivent refléter la vision d'une classe d'analystes [Kimball 1996] [Marcel 1998]. Cette vision correspond à une structuration des données selon plusieurs axes d'analyses (ou dimensions) pouvant représenter des notions variées telles que le temps, la localisation géographique, le code identifiant des produits… On parle alors de modélisation et de traitements multidimensionnels des données.

Les analyses décisionnelles sont basées sur des traitements OLAP directement reliés à la modélisation de l'information sous une forme conceptuelle proche de la perception qu'en a l'analyste. Cette perception de l'information est basée sur une vision multidimensionnelle des données.

Définition :
La **modélisation multidimensionnelle** consiste à considérer un sujet analysé comme un point dans un espace à plusieurs dimensions. Les données sont organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives de l'analyse.

Dans [Marcel 1998] l'auteur montre que la modélisation habituelle sous forme de relations devient inappropriée pour supporter la prise de décision et les analyses multidimensionnelles de type OLAP.

EXEMPLE : Considérons les données suivantes.

Ventes en 1999

Catégories des produits	Régions	Montant des ventes
Electroménager	Midi-Pyrénées	50
Electroménager	Aquitaine	40
Electroménager	Languedoc-Roussillon	30
Papeterie	Midi-Pyrénées	60
Papeterie	Languedoc-Roussillon	50
Bricolage	Midi-Pyrénées	30
Bricolage	Aquitaine	30

On peut distinguer différentes perspectives pour observer ces données : une dimension relative à la catégorie des produits, une dimension relative à la région.

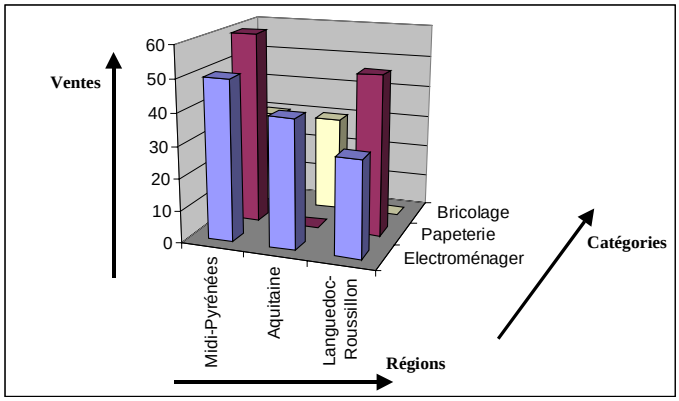


Figure 2 : Visualisation sous forme de barres.

Maintenant, nous considérons plusieurs tables, relatives aux ventes de chaque année entre 1997 et 1999. On peut alors observer les données dans un espace à trois dimensions : la dimension catégorie, la dimension produits et la dimension temps. Chaque intersection de ces dimensions représente une cellule comportant le montant des ventes.

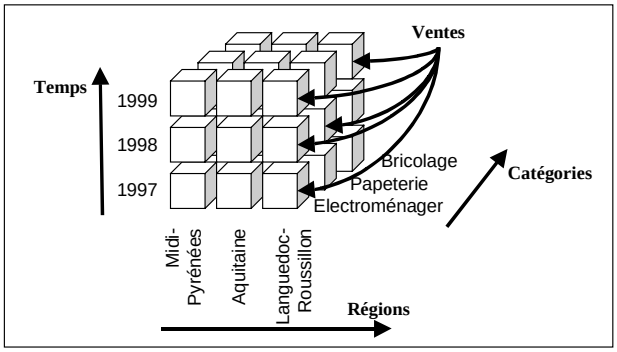


Figure 3 : Représentation multidimensionnelle.

2.4.1 Modélisation conceptuelle

Conceptuellement, cette modélisation multidimensionnelle a donné naissance aux concepts de fait et de dimension [Kimball 1996].

2.4.1.1 Concept de fait

Le sujet analysé est représenté par le concept de fait.

> *Définition* :
> Le **fait** modélise le sujet de l'analyse. Un fait est formé de **mesures** correspondant aux informations de l'activité analysée.

Les mesures d'un fait sont numériques et généralement valorisées de manière continue [Kimball 1996]. Les mesures sont numériques pour permettre de résumer un grand nombre d'enregistrements en quelques enregistrements (on peut les additionner, les dénombrer ou bien calculer le minimum, le maximum ou la moyenne). Les mesures sont valorisées de façon continue car il est important de ne pas valoriser le fait avec des valeurs nulles. Elles sont aussi souvent additives ou semi-additives afin de pouvoir les combiner au moyen d'opérateurs arithmétiques.

> EXEMPLE : Considérons le fait de *Vente* pouvant être constitué des mesures d'activités suivantes : quantité de produits vendus et montant total des ventes. Nous représenterons le fait par un rectangle englobant les différentes mesures d'activité qu'il contient. En outre le symbole d'un cube estampille le fait.

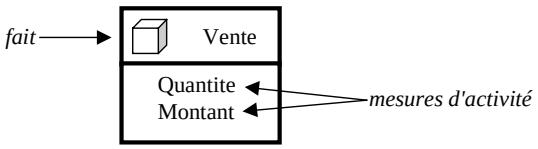


Figure 4 : Exemple de fait.

2.4.1.2 Concept de dimension

Le sujet analysé, c'est à dire le fait, est analysé suivant différentes perspectives. Ces perspectives correspondent à une catégorie utilisée pour caractériser les mesures d'activité analysées [Marcel 1998] ; on parle de dimensions.

> *Définition* :
> Une **dimension** modélise une perspective de l'analyse. Une dimension se compose de **paramètres** correspondant aux informations faisant varier les mesures de l'activité.

Les dimensions servent à enregistrer les valeurs pour lesquelles sont analysées les mesures de l'activité. Une dimension est généralement formée de paramètres (ou attributs) textuels et discrets. Les paramètres textuels sont utilisés pour restreindre la portée des requêtes afin de limiter la taille des réponses. Les paramètres sont discrets, c'est à dire que les valeurs possibles sont bien déterminées et sont des descripteurs constants.

> *EXEMPLE :* Poursuivons l'exemple précédent. Le fait peut être analysé suivant différentes perspectives correspondant à trois dimensions : la dimension *Temps*, la dimension *Geographie* et la dimension *Categorie*.
>
> Nous représenterons une dimension par un rectangle englobant les différents paramètres qu'elle contient. En outre un symbole représentant trois axes estampille les dimensions pour les distinguer du fait.

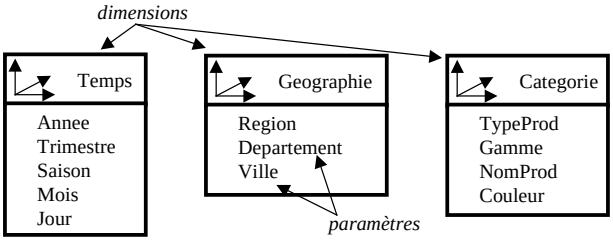


Figure 5 : Exemples de dimensions

Lors du processus OLAP, les données sont généralement analysées en partant d'un faible niveau de détail vers des données plus détaillées pour "*forer vers le bas*". Pour définir ces différents niveaux de détail, chaque dimension est munie d'une (ou plusieurs) hiérarchie(s) des paramètres. La hiérarchie sert lors des analyses pour restreindre ou accroître les niveaux de détail de l'analyse.

Définition :
Une **hiérarchie** organise les paramètres d'une dimension selon une relation "*est_plus_fin*" conformément à leur niveau de détail.

EXEMPLE : Les paramètres des dimensions sont organisés suivant une hiérarchie ; les paramètres sont ordonnés par une relation "*est_plus_fin*" et notée *P1* → *P2*. Par exemple, pour la dimension *Geographie* :

Ville → Departement → Region

Ainsi chaque ville appartient à un département qui est situé dans une région.

2.4.1.3 Modèles en étoile, en flocon et en constellation

A partir du fait et des dimensions, il est possible d'établir une structure de données simple qui correspond au besoin de la modélisation multidimensionnelle. Cette structure est constituée du fait central et des dimensions. Ce modèle représente visuellement une étoile, on parle de **modèle en étoile** (*star schema* [Kimball 1996]).

EXEMPLE : La Figure 6 décrit le schéma en étoile modélisant les analyses des quantités et des montants des médicaments dans les pharmacies selon trois dimensions : le temps, la catégorie et la situation géographique.

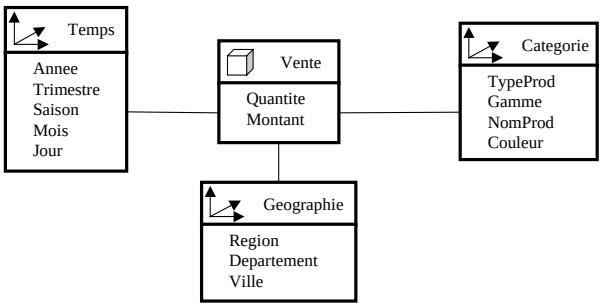


Figure 6 : Exemple d'une modélisation en étoile.

Il existe d'autres techniques de modélisation multidimensionnelle, notamment la **modélisation en flocon** (*snowflake*). Une modélisation en flocon consiste à décomposer les dimensions du modèle en étoile en sous hiérarchies. La modélisation en flocon est donc une émanation de la

modélisation en étoile ; le fait est conservé et les dimensions sont éclatées conformément à sa hiérarchie des paramètres.

L'avantage de cette modélisation est de formaliser une hiérarchie au sein d'une dimension. Par contre, la modélisation en flocon induit une dénormalisation des dimensions générant une plus grande complexité en termes de lisibilité et de gestion.

EXEMPLE : La Figure 7 illustre la modélisation en flocon ; nous décrivons le modèle en étoile de la Figure 6 en dénormalisant chacune de ces dimensions, formant ainsi une sorte de flocon.

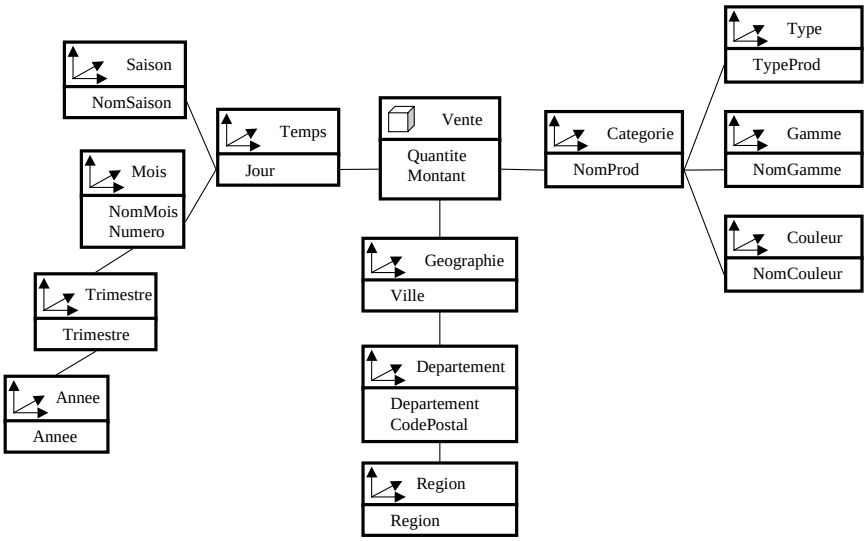


Figure 7 : Exemple d'une modélisation en flocon.

Une autre technique de modélisation, issue du modèle en étoile, est la **modélisation en constellation**. Il s'agit de fusionner plusieurs modèles en étoile qui utilisent des dimensions communes. Un modèle en constellation comprend donc plusieurs faits et des dimensions communes ou non.

EXEMPLE : La Figure 8 illustre la modélisation en constellation ; nous décrivons une constellation constituée de deux schémas en étoile : l'un correspond aux ventes effectuées dans les pharmacies et l'autre analyse les prescriptions des médecins.

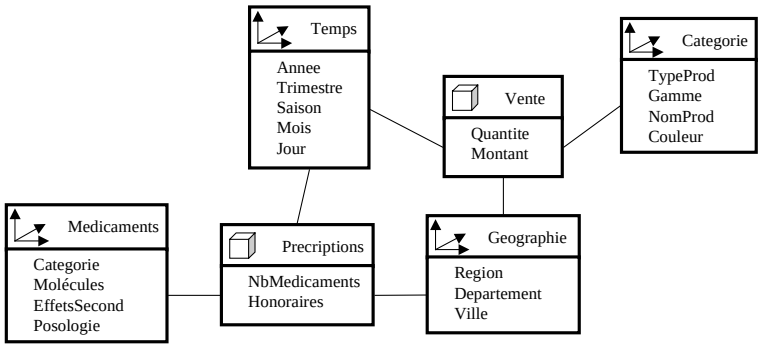


Figure 8 : Exemple d'une modélisation en constellation.

Les dimensions *Temps* et *Geographie* sont partagées par les faits *Prescriptions* et *Vente*.

2.4.2 Modélisation logique

Au niveau logique plusieurs possibilités sont envisageables pour la modélisation multidimensionnelle. Il est possible d'utiliser :

- un système de gestion de bases de données (SGBD) existant tel que les SGBD relationnels (ROLAP) ou bien les SGBD orientées objet (OOLAP),
- un système de gestion de bases de données multidimensionnelles (MOLAP).

2.4.2.1 ROLAP et OOLAP

L'approche la plus couramment utilisée consiste à utiliser un système de gestion de bases de données relationnelles, on parle de l'approche **ROLAP** ("*Relational On-Line Analytical Processing*"). Le modèle muldimensionnel est alors traduit de la manière suivante :

- chaque fait correspond à une table, appelée **table de fait**,
- chaque dimension correspond à une table, appelée **table de dimension**.

Ainsi, la table de fait est constituée d'attributs représentant les mesures d'activité et les attributs clés étrangères de chacune des tables de dimension. Les tables de dimension contiennent les paramètres et une clé primaire permettant de réaliser des jointures avec la table de fait.

> *EXEMPLE :* Nous considérons l'exemple du schéma en étoile de la Figure 6. La table *VENTE* correspond au fait et les tables *TEMPS*, *GEOGRAPHIE*, *CATEGORIE* représentent les dimensions.
>
> **VENTE**(CleTps#, CleGeo#, CleCat#, Quantite, Montant)
> **TEMPS**(CleTps, Annee, Trimestre, Saison, Mois, Jour)
> **GEOGRAPHIE**(CleGeo, Region, Departement, Ville)
> **CATEGORIE**(CleCat, TypeProd, Gamme, NomProd, Couleur)

Plus récemment, une autre approche s'appuie sur le paradigme objet ; on parle de l'approche **OOLAP** ("*Object On-Line Analytical Processing*"). Le modèle multidimensionnel se traduit ainsi :

- chaque fait correspond à une classe, appelée **classe de fait**,
- chaque dimension correspond à une classe, appelée **classe de dimension**.

> *EXEMPLE :* Nous considérons l'exemple du schéma en étoile de la Figure 6. La classe *Vente* correspond au fait et les classes *Temps*, *Geographie*, *Categorie* représentent les dimensions. Les expressions suivantes décrivent le schéma en étoile en utilisant le langage de définition standard des bases de données orientées objet défini par l'ODMG[1] [Cattel 1995].

```
interface Vente {
   attribute Integer quantite;
   attribute Double montant;
   relationship Set<Temps> dim_temps inverse Temps::fait_vente;
   relationship Set<Geographie> dim_geo inverse Geographie::fait_vente;
   relationship Set<Categorie> dim_categ inverse Categorie::fait_vente;   }
```

[1] http://www.odmg.org/

```
interface Temps {
   attribute Integer annee;
   attribute String trimestre;
   attribute String saison;
   attribute String mois;
   attribute Integer jour;
   relationship Vente fait_vente inverse Vente::dim_temps;   }

interface Geographie {
   attribute String region;
   attribute String departement;
   attribute String ville;
   relationship Vente fait_vente inverse Vente::dim_geo;   }

interface Categorie {
   attribute String type_prod;
   attribute String gamme;
   attribute String nom_prod;
   attribute Set<String> couleur;
   relationship Vente fait_vente inverse Vente::dim_categ;   }
```

Dans [Pedersen, Jensen 1999], les auteurs identifient 9 critères indispensables que doivent supporter les modèles multidimensionnels. Plusieurs de ces critères peuvent être simplement modélisés avec l'approche OOLAP, tels que le besoin d'expliciter les hiérarchies des dimensions, de supporter correctement l'agrégation et les relations multivaluées entre le fait et les dimensions, de gérer et manipuler simplement le temps. L'avantage de l'approche OOLAP par rapport à l'approche ROLAP est donc liée au niveau d'abstraction plus fort de l'objet qui prend en compte des concepts plus riches [Cauvet, Semmak 1994], permettant ainsi de modéliser facilement certaines caractéristiques des données multidimensionnelles [Pedersen, Jensen 1998].

2.4.2.2 MOLAP

Une alternative à ces deux approches consiste à utiliser un système multidimensionnel "*pur*" qui gère des structures multidimensionnelles natives ; on parle de l'approche **MOLAP** ("*Multidimensional On-Line Analytical Processing*").

Les structures multidimensionnelles natives utilisées sont des tableaux à *n* dimensions. Dans la littérature, les termes de cube, hypercube et table multidimensionnelle sont utilisés de manière interchangeable. Nous utiliserons le terme d'**hypercube** pour désigner des structures à deux, trois ou à plus de trois dimensions.

Cette approche permet de stocker les données de manière multidimensionnelle. L'intérêt est que les temps d'accès sont optimisés, mais cette approche nécessite de redéfinir des opérations pour manipuler ces structures multidimensionnelles.

2.5 Synthèse

Nous avons défini le **système décisionnel** comme un système d'information dédié aux applications décisionnelles qui comprend essentiellement deux types de composants : un entrepôt de données et des magasins de données.

L'**entrepôt de données** doit collecter l'ensemble de l'information utile aux décideurs à partir des sources de données. L'objectif de l'entrepôt est de centraliser l'information décisionnelle en assurant l'intégration des données extraites, la pérennité de ces données dans le temps et la conservation de leurs évolutions.

Les **magasins de données** consistent à extraire une partie de l'information décisionnelle contenue dans l'entrepôt ; il s'agit de la partie des données utile pour une classe d'utilisateurs ou pour un besoin d'analyse spécifique, en ce sens ils sont orientés sujet. L'objectif du magasin est de supporter efficacement des processus d'analyse de type OLAP.

Ainsi, les données stockées dans un magasin doivent correspondre à une structuration adaptée des données (selon plusieurs axes d'analyses) reflétant la vision des analystes. Cette représentation des données est basée sur une **modélisation multidimensionnelle.** L'approche multidimensionnelle vise à modéliser les données conformément à la perception des analystes : les données sont représentées suivant les différents axes d'analyses possibles. Le modèle multidimensionnel comprend un **fait** contenant les mesures à analyser et des **dimensions** contenant les paramètres de l'analyse. Dans chaque dimension, les paramètres sont hiérarchisés selon des niveaux de détail.

3 ETAT DE L'ART SUR LES ENTREPOTS ET LES MAGASINS DE DONNEES

Cette section présente différents axes de recherches relatifs aux entrepôts et magasins de données. Nous décrivons également les grands projets de recherche dans le domaine des entrepôts et nous présentons l'offre commerciale d'*Oracle Express* qui nous semble représentative des produits OLAP sur le marché.

3.1 Différents axes de recherches

Issus originellement de l'industrie, les entrepôts et magasins de données sont devenus aujourd'hui un thème de recherche[2] à part entière [Widom 1995] [Chaudhuri, Dayal 1997]. Nous avons retenu deux axes de recherche principaux :

- l'extraction et le stockage des données,
- la modélisation des entrepôts et des magasins ainsi que leur langage d'interrogation.

[2] *Bibliographies disponibles sur Internet :*
http://www.cs.toronto.edu/~mendel/dwbib.html,
http://www.informatik.tu-darmstadt.de/DVS1/staff/wu/dw.html,
http://www-db.stanford.edu/warehousing/publications.html.

L'**extraction des données** repose essentiellement sur la technique des vues matérialisées [Gupta, Mumick 1995] [Widom 1995]. Une vue matérialisée consiste à calculer une vue exprimée sur une source de données et à stocker physiquement les données obtenues dans l'entrepôt. Cette approche induit des problématiques de sélection et de maintenance des vues matérialisées. En effet, il faut définir des algorithmes qui répercutent les évolutions des données source au niveau des copies de données contenues dans l'entrepôt. Mais il faut aussi définir des algorithmes qui calculent un ensemble optimal de vues à matérialiser de telle sorte que les coûts liés à la maintenance de ces vues ne viennent pas altérer le fonctionnement de l'entrepôt.

Le but de **la modélisation des entrepôts et des magasins** de données est de fournir des abstractions permettant de détacher la manière de représenter les données de leur implantation physique. L'utilisation qui est faite des systèmes décisionnels (analyses décisionnelles au travers de processus OLAP) nécessite des représentations des données différentes de celles qui sont proposées dans les bases de données classiques [Codd 1993] [Kimball 1996]. Il faut organiser les données décisionnelles en fonction des analyses multidimensionnelles effectuées (analyses des données suivants plusieurs axes). D'autre part, la modélisation des données doit prendre en compte leur aspect évolutif (historisation des données) ; en effet, par nature, l'entrepôt de données conserve l'historique des informations décisionnelles [Inmon 1994] [Chaudhuri, Dayal 1997]. En outre, ces nouvelles représentations induisent de nouveaux besoins en terme de manipulation et d'interrogation des données ; ceci nécessite d'étendre les langages d'interrogation actuels [RedBrick 1998].

L'objet de cette thèse concerne plus particulièrement la modélisation des entrepôts (élaborés à partir de sources de données) **et des magasins de données** (construits à partir d'un entrepôt). Dans la suite de cette section, nous effectuons un état de l'art dans le domaine des entrepôts où nous étudions les travaux relatifs à :

- l'extraction au travers de la technique des vues matérialisées et
- la modélisation qui le plus souvent fait appel à une modélisation multidimensionnelle.

3.2 Travaux sur les vues matérialisées

La technique des vues matérialisées est couramment utilisée pour définir les données copiées dans l'entrepôt [Widom 1995] [Gupta, Mumick 1995] [Chaudhuri, Dayal 1997]. De nombreux travaux traitent des problématiques concernant les vues matérialisées dans le contexte des entrepôts de données. Nous pouvons distinguer deux thèmes de recherche principaux :

- La **maintenance incrémentale** des vues matérialisées qui se propose de répercuter immédiatement les mises à jour survenues au niveau des sources de données.
- La **configuration** de l'entrepôt (sélection des vues à matérialiser) qui se propose de déterminer un ensemble de vues à matérialiser dans l'entrepôt de telle sorte que le coût de maintenance soit optimal.

Ces travaux abordent les systèmes décisionnels sans distinguer l'espace de stockage en un entrepôt de données et des magasins de données ; ils représentent le système décisionnel comportant un unique espace de stockage appelé entrepôt. Dans les sous-sections suivantes nous utiliserons le terme entrepôt comme un terme générique désignant l'espace de stockage du système décisionnel.

3.2.1 Recherches sur la maintenance incrémentale

Nous présentons les caractéristiques ainsi qu'un comparatif des travaux relatifs à la maintenance incrémentale des vues matérialisées dans le contexte des entrepôts de données.

Le Tableau 2 présente les caractéristiques essentielles des travaux sur la maintenance.

Travaux	Caractéristiques
[Gupta, Mumick 1995] **[Quass, et al 1996]**	- taxonomie des problèmes relatifs aux vues matérialisées, - calcul de vues auxiliaires permettant l'auto-maintenance.
[Hull, Zhou 1996] **[Zhou, et al 1996]**	- combinaison de vues virtuelles, partiellement matérialisées ou matérialisées.
[Hurtado, et al 1999]	- algorithme de maintenance qui tient compte des structures multidimensionnelles.
[Huyn 1996, 1997]	- définition de critères déterminant les vues auto-maintenables, - algorithme de maintenance avec un accès partiel aux sources.
[Labio, et al 1999]	- algorithme de maintenance d'un ensemble de vues.
[Labio, Garcia-Molina 1996]	- algorithme réduisant les coûts de maintenance et le volume matérialisé par des opérateurs de jointures et semi-jointures.
[Mumick, et al 1997]	- algorithme de maintenance pour des vues intégrant des fonctions d'agrégations.
[Quass, Widom 1997]	- algorithme de maintenance concurrente avec les transactions de la source et l'interrogation de l'entrepôt.
[Yang, Widom 1998, 2000]	- vues intégrant des opérateurs temporels (sous-ensemble de TSQL2), - algorithmes de maintenance et d'auto-maintenance des vues temporelles.
[Zhuge, et al 1995, 1996, 1997, 1998]	- définition de différents niveaux de consistance entre les vues de l'entrepôt et les données source, - familles d'algorithmes associées à chaque niveau de consistance.
[Zhuge, Garcia-Molina 1998]	- extension des travaux sur la maintenance incrémentale de [Zhuge, et al 1998] par un graphe d'objets (limité à des structures simples).

Tableau 2 : Caractéristiques des travaux sur la maintenance incrémentale des vues matérialisées dans le domaine des entrepôts de données.

Le Tableau 3 effectue une comparaison des travaux sur la maintenance incrémentale.

- La première colonne indique le modèle de données utilisé dans l'entrepôt. Plusieurs modèles peuvent être utilisés : les modèles habituels dans les bases de données comme le modèle relationnel (*R*) et le modèle objet (*O*), ou bien des modèles spécifiques aux systèmes décisionnels tel que les modèles multidimensionnels (*M*).
- La seconde colonne décrit les vues utilisées. Il s'agit de vues pouvant être :
  - virtuelles, notées *V* (les données de la vue restent physiquement stockées au niveau des sources et la vue est calculée au moment de l'interrogation),
  - matérialisées, notées *M* (la vue est calculée avant l'interrogation et les données sont physiquement stockées dans l'entrepôt),
  - auxiliaires, notées *A* (une vue auxiliaire est une vue, généralement matérialisée, non directement définie par l'administrateur ; elle est utilisée par le système pour améliorer le fonctionnement de l'entrepôt en conservant des informations supplémentaires).

  D'autre part, la définition des vues peut être effectuée au travers d'opérateurs de sélection, de projection et de jointure (*SPJ*), d'opérateurs de groupement associés à des agrégations et enfin d'opérateurs temporels.

- La troisième colonne indique les techniques mises en œuvre. Les travaux s'appuient sur :
  - des modèles de graphe (ils sont utilisés pour décrire les différents scénarios de calcul d'une vue appelés plan de décomposition d'une vue),
  - des modèles de coûts (ils servent à modéliser les coûts liés au calcul d'une vue ou à sa maintenance),
  - des stratégies d'auto-maintenance des vues (il s'agit maintenir la vue sans accéder ou en limitant l'accès aux sources notamment par l'ajout de vues auxiliaires).

	Modèle de données	**Vues**				**Techniques utilisées**
		Types des vues	**SPJ**	**Agrégations et groupements**	**Opérateurs temporels**	
[Gupta, Mumick 1995] [Quass, et al 1996]	R	M/A	x			Modèle de graphe, Stratégie d'auto-maintenance.
[Hull, Zhou 1996] [Zhou, et al 1996]	R	M/V	x			Modèle de graphe.
[Hurtado, et al 1999]	M	M	x	x		Modèle de graphe.
[Huyn 1996, 1997]	R	M	x			Modèle de graphe, Stratégie d'auto-maintenance.
[Labio, et al 1999]	R	M	x	x		Modèle de graphe.
[Labio, Garcia-Molina 1996]	R	M	x			Modèle de coût
[Mumick, et al 1997]	R(*)	M/A	x	x		Modèle de graphe.
[Quass, Widom 1997]	R	M	x	x		
[Yang, Widom 1998, 2000]	R	M	x		x	Stratégie d'auto-maintenance.
[Zhuge, et al 1995, 1996, 1997, 1998]	R	M	x			
[Zhuge, Garcia-Molina 1998]	O	M	x			Modèle de graphe.

Tableau 3 : Comparaison des travaux sur la maintenance incrémentale des vues matérialisées dans le domaine des entrepôts de données.

(*) Ces travaux n'utilisent pas un modèle multidimensionnel mais prennent en compte la structure multidimensionnelle du système décisionnel.

Les deux tableaux précédents, décrivant les travaux de recherche sur la maintenance des vues matérialisées, indiquent que les travaux actuels se focalisent sur des vues relationnelles.

[Zhuge, et al 1998] constitue une étude complète, dans laquelle différents algorithmes permettent de maintenir de manière incrémentale des vues relationnelles SPJ en fonction de différents niveaux de consistance (entre l'entrepôt et les sources de données) :

- la convergence ("*Convergence*") garantit qu'après la dernière transaction, l'entrepôt est consistant avec la source,
- la faible consistance ("*Weak consistency*") garantit l'existence de transactions où chaque état de l'entrepôt reflète un état valide des sources. Cependant, l'entrepôt peut générer des transactions différentes de la source.
- la forte consistance ("*Strong consistency*") garantit l'existence de transactions où chaque état de l'entrepôt reflète un état valide des sources. L'entrepôt génère des transactions correspondantes aux transactions des sources.

- la complétude ("*Completeness*") garantit en plus de la forte consistance, la préservation de l'ordre entre les états de la vue de l'entrepôt et les états des sources.

Les techniques de maintenance proposées sont incrémentales (les mises à jour de la source sont répercutées immédiatement). Certains travaux traitent de l'approche d'auto-maintenance qui consiste à maintenir la vue en limitant les accès aux sources, notamment en utilisant des vues auxiliaires.

Les vues sont, de manière générale, définies au travers d'opérations de sélection, de projection et de jointure. Peu de travaux utilisent les agrégations [Mumick, et al 1997] et des opérateurs temporels [Yang, Widom 1998, 2000] qui sont pourtant essentiels dans les entrepôts de données, respectivement pour résumer l'information et pour historiser les données.

3.2.2 Recherches sur la configuration

Nous présentons les caractéristiques ainsi qu'une comparaison des travaux relatifs à la configuration de l'entrepôt, c'est à dire à la détermination des vues à matérialiser.

Travaux	Caractéristiques
[Baralis, et al 1997]	- algorithme de sélection des vues à matérialiser tenant compte de la structure multidimensionnelle, - prise en compte des temps de réponse à l'interrogation.
[Gupta 1997] **[Gupta, Mumick 1999]**	- algorithmes gloutons de calcul des vues à matérialiser basé sur des heuristiques, - prise en compte de la limitation d'espace de stockage et du temps de calcul pour l'ensemble à matérialiser.
[Harinarayan, et al 1996]	- algorithme glouton de calcul des vues à matérialiser, - prise en compte du coût de maintenance, de la limitation d'espace de stockage et de temps de calcul.
[Kotidis, Roussopoulos 1999]	- algorithme de calcul des vues à matérialiser, - adaptation dynamique (recalcul) de cet ensemble en fonction de l'usage de l'entrepôt, - prise en compte de contraintes de volume stockage.
[Labio, et al 1997]	- algorithme calculant un ensemble optimal de vues et d'index, - prise en compte du coût de maintenance avec réduction de l'espace de stockage.
[Shukla, et al 1998]	- algorithme de calcul des vues à matérialiser, tenant compte des structures multidimensionnelles.
[Theodoratos, Sellis 1997, 1999]	- algorithme calculant un ensemble optimal de vues, - compromis entre le coût de maintenance et les performances de l'interrogation.
[Yang, et al 1997]	- algorithmes de sélection des vues à matérialiser utilisant des heuristiques, - prise en compte du coût de maintenance et des performances de l'interrogation des vues.

Tableau 4 : Caractéristiques des travaux sur la sélection des vues à matérialiser dans les entrepôts de données.

Le Tableau 4 décrit les caractéristiques des travaux abordant la problématique de la sélection d'un ensemble optimal de vues à matérialiser en fonction de critères de limitation d'espace de stockage, de temps de calcul, de coût de maintenance.

Le Tableau 5 compare ces travaux de recherche sur la sélection des vues à matérialiser. Nous considérons les mêmes critères de comparaison que dans le Tableau 3.

	Modèle de données	Vues				Techniques utilisées
		Types des vues	SPJ	Agrégations et groupements	Opérateurs temporels	
[Baralis, et al 1997]	M	M	x	x		Modèle de graphe, Modèle de coût.
[Gupta 1997] **[Gupta, Mumick 1999]**	R	M	x			Modèle de graphe, Modèle de coût.
[Harinarayan, et al 1996]	M	M	x	x		Modèle de graphe, Modèle de coût.
[Kotidis, Roussopoulos 1999]	R	M	x	x		Modèle de graphe, Modèle de coût, Stratégie d'auto-maintenance.
[Labio, et al 1997]	R	M	x	x		Modèle de graphe, Modèle de coût.
[Shukla, et al 1998]	M	M	x	x		Modèle de graphe, Modèle de coût.
[Theodoratos, Sellis 1997, 1999]	R	M	x			Modèle de graphe, Modèle de coût.
[Yang, et al 1997]	R	M	x	x		Modèle de graphe, Modèle de coût.

Tableau 5 : Comparaison des travaux sur la sélection des vues à matérialiser dans les entrepôts de données.

On constate que les travaux se placent dans un contexte d'entrepôts relationnels. Les vues sont définies au travers d'opérations de sélection, de projection et de jointure ainsi que des fonctions d'agrégations. Tous ces travaux se basent sur des techniques similaires de modèles de graphe et de modèles de coûts.

Cependant, [Kotidis, Roussopoulos 1999] se distingue en proposant de sélectionner dynamiquement l'ensemble optimal des vues à matérialiser. L'originalité du système *Dynamat* proposé réside dans l'évolution permanente de l'ensemble des vues sélectionnées (pour être matérialisées) en fonction de l'utilisation de l'entrepôt (requêtes) mais également de contraintes liés au volume des données matérialisées. En effet, les autres approches, se contentent de calculer un ensemble optimal de vues à matérialiser, mais ne permettent pas son évolution dans le temps.

3.2.3 Travaux connexes

Quelques travaux étudient des problèmes particuliers liés aux vues matérialisées dans le contexte des entrepôts. Ces points concernent principalement la gestion des données, c'est à dire :

- le traitement des données qui expirent (il s'agit de données qui deviennent inutiles dans l'entrepôt ou inadaptées sous leur forme actuelle) [Garcia-Molina, et al 1998],
- le traitement des relations entre les données de l'entrepôt et les données source d'où sont issues celles de l'entrepôt [Cui, Widom 2000],
- la gestion de la définition des vues matérialisées en fonction de l'évolution du schéma des sources [Bellahsene 1998].

[Garcia-Molina, et al 1998] aborde le problème de l'expiration des données matérialisées dans l'entrepôt. L'approche proposée consiste simplement à supprimer l'information devenue obsolète. Cette approche est donc limitée car elle ne propose pas de mécanismes plus souples comme l'archivage de certaines données. Une des contributions de nos travaux présentés dans cette thèse est justement la proposition d'un mécanisme automatique permettant d'archiver les données dont le détail est devenu obsolète ; l'administrateur définit les critères de détermination des données expirées ainsi que l'opération d'archivage appliquée sur ces données.

[Cui, Widom 2000] étudie les problèmes d'identification des données source à partir desquelles sont construites les données matérialisées de l'entrepôt (*Tracing Lineage Problem*). L'approche proposée intègre des vues relationnelles SPJ avec des agrégations et s'appuie sur des vues auxiliaires permettant de limiter l'accès aux relations source. Dans notre approche, nous proposons de maintenir une trace permettant de retrouver les données source d'origine ; ceci est essentiel pour répercuter les évolutions des sources tout en maintenant l'historique des données.

[Bellahsene 1998] propose un environnement permettant de maintenir la définition des vues matérialisées dans un entrepôt en fonction des évolutions de schémas qui surviennent au niveau des relations source. Cette approche consiste à étendre le langage de définition des vues (SQL) pour indiquer des priorités sur les attributs impliqués dans la définition d'une vue : les attributs peuvent être indispensables (leur suppression nécessite de redéfinir la vue), ou bien remplacés par un (ou plusieurs) attribut(s) équivalent(s).

3.3 Travaux sur l'approche multidimensionnelle

Un deuxième axe de recherche important dans le domaine des entrepôts de données concerne l'approche multidimensionnelle.

Dans l'architecture des systèmes décisionnels, la modélisation multidimensionnelle concerne plus particulièrement les magasins de données. En effet, l'approche multidimensionnelle représente les données conformément aux traitements effectués par les décideurs, suivant les différents axes d'analyses possibles. Le modèle multidimensionnel se compose de **faits** contenant les mesures à analyser et de **dimensions** contenant les paramètres de l'analyse. Dans chaque dimension, les paramètres sont organisés hiérarchiquement en niveaux de détail.

Cette représentation multidimensionnelle des données induit de nouvelles opérations liées à l'aspect "*analyse*". Ces opérations élémentaires de manipulation sont liées aux structures et au niveau d'observation des valeurs, appelé niveau de granularité des données.

3.3.1 Manipulation des données multidimensionnelles

3.3.1.1 Visualisation des données multidimensionnelles

Les données multidimensionnelles sont visualisées sous une forme reflétant leur caractère multidimensionnel. Plusieurs possibilités sont offertes parmi lesquelles la représentation sous forme de tableau est la plus intuitive et la plus courante (*cf.* Figure 9). Une table permet de représenter toutes les combinaisons des valeurs choisies pour constituer les noms de lignes et les noms de colonnes. Ces combinaisons peuvent ne pas être connues ; le symbole *null* représente cette absence.

Régions →

Catégories ↓

Montant des ventes en 1999	Midi-Pyrénées	Aquitaine	Languedoc-Roussillon
Electroménager	50	40	30
Papeterie	60	*null*	50
Bricolage	30	30	*null*

Figure 9 : Exemple d'une table représentant le montant des ventes en 1999.

Il est possible de visualiser sous la forme d'une table des données enregistrées dans des relations (ROLAP), des classes (OOLAP) ou des structures multidimensionnelles (MOLAP). Cependant, lorsque le nombre de dimensions est supérieur à deux, l'utilisateur ne peut visualiser simultanément l'ensemble de l'information. Il doit disposer d'opérations suffisamment puissantes pour manipuler les données ; par exemple permuter les dimensions ou les plans (pour visualiser l'information suivant les différentes dimensions).

Dans les sous-sections suivantes, nous décrivons les opérations attachées à la manipulation des données multidimensionnelles.

3.3.1.2 Opérations classiques

Parmi les opérations classiques [Chrisment, et al 1999] retenues (ces opérations correspondent aux opérations relationnelles de manipulation des données), nous pouvons citer :

- la **sélection** (*slice and dice* ou *select* en relationnel),
- la **projection**,
- le **produit cartésien** et la **jointure**,
- les opérations ensemblistes d'**union**, d'**intersection** et de **différence**,
- la **suppression** d'une mesure, l'**ajout** d'une mesure calculée et le **renommage**.

3.3.1.3 Opérations agissant sur la structure

Les opérations agissant sur la structure multidimensionnelle visent à changer le point de vue des données observées.

- La **rotation** (*rotate*) consiste à effectuer une rotation de l'hypercube, de manière à présenter une face différente.
- La **permutation** (*switch*) consiste à inverser deux dimensions, de manière à permuter deux tranches de l'hypercube.
- La **division** (*split*) consiste à représenter chaque tranche de l'hypercube en passant à une représentation tabulaire. Plus généralement, cette opération permet de réduire le nombre de dimensions.
- L'**emboîtement** (*nest*) permet d'imbriquer les positions (valeurs) d'un paramètre d'une dimension avec un autre paramètre. L'intérêt de cette opération est qu'elle permet de représenter de manière bidimensionnelle toutes les informations de l'hypercube quelque soit le nombre de dimensions.

- L'**enfoncement** (*push*) consiste à combiner les positions (valeurs) d'un paramètre d'une dimension aux mesures du fait et donc de transformer un paramètre en mesure. L'opération inverse de **retrait** (*pull*) permet de transformer une mesure en paramètre en changeant le statut de certaines mesures de l'hypercube pour constituer une nouvelle dimension.
- La **factualisation** (*fold*) consiste à transformer une dimension en mesure(s) ; cette opération permet de transformer en mesure l'ensemble des paramètres d'une dimension. La **paramétrisation** (*unfold*) permet de transformer une mesure en paramètre dans une nouvelle dimension.
- L'opération **Cube** permet de calculer des sous-totaux et un total final dans l'hypercube.

3.3.1.4 Opérations agissant sur la granularité

Les opérations agissant sur la granularité d'observation des données caractérisent la hiérarchie de navigation entre les différents niveaux. Elles correspondent aux opérations suivantes :

- Le **forage vers le haut** (*drill-up* ou *roll-up* ou *scale-up*) consiste à représenter les données de l'hypercube à un niveau de granularité supérieur conformément à la hiérarchie définie sur la dimension. Une fonction d'agrégation (somme, moyenne…) en paramètre de l'opération indique comment sont calculées les valeurs du niveau supérieur à partir de celles du niveau inférieur.
- Le **forage vers le bas** (*drill-down* ou *roll-down* ou *scale-down*) consiste à représenter les données de l'hypercube à un niveau de granularité inférieur, donc sous une forme plus détaillée.

Le forage vers le haut ou vers le bas nécessite des informations non contenues dans l'hypercube pour passer d'une représentation initiale à une représentation de granularité différente. L'augmentation nécessite de connaître la fonction d'agrégation utilisée tandis que l'affinement nécessite de connaître les données au niveau inférieur.

3.3.2 Recherches sur la modélisation multidimensionnelle

Le Tableau 6 présente une étude comparative des modèles multidimensionnels actuellement proposés dans la littérature. Ces modèles sont des modèles MOLAP, ROLAP ou OOLAP manipulant les structures de données suivantes : tableaux à n dimensions (*T*), relations (*R*) et objets (*O*).

En plus du modèle utilisé, nous considérons les caractéristiques suivantes :

- la correspondance avec le modèle relationnel permettant d'exprimer plus simplement certaines opérations comme les opérations classiques des bases de données,
- la représentation explicite des hiérarchies (de niveaux de détail) sur les dimensions,
- la possibilité de manipuler à la fois le contenu et la structure des données,
- la prise en compte du temps de manière spécifique,
- le type de langage utilisé pouvant être de type algèbre (*Al*), calcul (*Ca*), règles (*Re*),
- les opérations associées au modèle de données.

L'ensemble des opérations, associées au modèle multidimensionnel, permet de manipuler les données et les structures de données. Ces opérations visent à aider l'utilisateur pour l'analyse des données suivant différentes perspectives. Le tableau précédent met en évidence l'absence d'un modèle de données offrant toutes les opérations.

		[Li, Wang 1996]	**[Cabbibo, Torlone 1997, 1998]**	**[Gyssen, Lakshmanan 1997]]**	**[Agrawal, et al 1997]]**	**[Marcel 1998]]**	**[Lehner, et al 1998] [Lehner 1998]**	**[Pedersen, Jensen 1999]**	**[Mendelzon, Vaisman 2000]**
Modèle de données		T, R	T	T	T	T	O	O	T
Correspondance relationnelle			x	x	x	x		x	x
Hiérarchies des dimensions		x	x			x		x	x
Contenu et structure			x	x	x	x			x
Gestion du temps								x	x
Type de langage		Ca	Re	Al	Al	Re	Al	Al	Re
Opérations	*Renommage*	x	x	x		x		x	
	Ajout de mesures		x			x			
	Suppression					x			
	Sélection		x	x	x	x	x	x	x
	Jointure	x	x	x	x			x	x
	Union, Différence, Intersection	x	x	x	x			x	
	Agrégation	x	x	x	x		x	x	x
	Roll-up	x	x	x	x	x	x	x	x
	Drill-down						x	x	x
	Cube			x		x			
	Switch			x					
	Push				x	x			
	Pull				x	x			
	Fold		x	x					
	Unfold		x	x					
	Rotate						x		
	Nest						x		
	Split			x			x		x

Tableau 6 : Comparatif des travaux sur la modélisation multidimensionnelle.

La dimension temporelle des données est généralement considérée comme une dimension quelconque. Cette approche ne permet pas d'exprimer des requêtes temporelles complexes comparables aux possibilités offertes dans les bases de données temporelles. Face à cette limite, [Pedersen, Jensen 1999] aborde la modélisation temporelle dans les entrepôts multidimensionnels. Néanmoins, l'évolution des données n'est pas prise en compte (historisation des données). Dernièrement, [Mendelzon, Vaisman 2000] intègre partiellement, dans un schéma multidimensionnel, des mécanismes de manipulation correspondant à ceux définis dans les bases de données temporelles. Il s'agit d'une modélisation multidimensionnelle temporelle (définition spécifique de la dimension temps) associée au langage TOLAP qui supporte des opérations temporelles (sous-ensemble de TSQL2).

Toutefois, aucune de ces propositions ne gère l'archivage des données, c'est à dire la possibilité de faire évoluer dynamiquement dans l'entrepôt la granularité temporelle des données ; un tel mécanisme permet à l'administrateur de conserver dans l'entrepôt les données temporelles sous une forme adaptée aux besoins des décideurs et de faire évoluer les données de l'entrepôt en fonction leur ancienneté.

En outre, les travaux étudiant des solutions permettant d'organiser les données de manière multidimensionnelle sont peu nombreux et restent empiriques. A notre connaissance, une seule étude propose une démarche de transformation d'un schéma E/A en un schéma en étoile [Golfarelli, et al 1998] [Golfarelli, Rizzi 1999].

3.4 Grands projets de recherche

Nous décrivons les principaux projets de recherche sur les entrepôts de données, en décrivant l'objectif de chacun et le fonctionnement du système développé.

3.4.1 DWQ[3]

DWQ (*Foundations of Data Warehouse Quality*) [Jarke, et al 1998] [Jeusfeld, et al 1998] est un projet de la communauté Européenne visant à développer des fondements sémantiques qui permettront d'aider les concepteurs d'entrepôts de données dans le choix des modèles, des structures de données avancées et des techniques d'implantation efficaces en s'appuyant sur des facteurs de qualité de service. Ceci permet d'améliorer la conception, l'exploitation et l'évolution des applications d'entrepôts.

DWQ s'appuie sur des modèles formels pour la qualité. Les résultats comportent des méta modèles de données formels destinés à la description de l'architecture statique d'un entrepôt de données. Les outils associés comportent des facilités de modélisation incluant des caractéristiques spécifiques aux entrepôts comme la résolution de sources multiples, la gestion de données multidimensionnelles (éventuellement agrégées) et des techniques pour l'optimisation de requêtes et la propagation incrémentale des mises à jour.

3.4.2 SIRIUS[4]

Le projet SIRIUS (*Supporting the Incremental Refreshment of Information warehoUseS*) [Vavouras, et al 1999] développé à l'Université de Zurich, est un système d'entrepôt de données qui a pour objectif d'étudier des techniques permettant le rafraîchissement incrémental de l'entrepôt en réduisant les temps de mise à jour. Le schéma de l'entrepôt est défini sous la forme d'un schéma global UML.

Les sources opérationnelles sont équipées d'un traducteur et d'un contrôleur qui permettent la communication entre le modèle de la source et un modèle commun, interne au système. Le contrôleur permet en particulier de détecter les évolutions de la source à laquelle il est associé. Le module de gestion du rafraîchissement est le module central. Il est muni d'un contrôleur responsable de la gestion du processus de rafraîchissement et d'un ensemble de médiateurs spécialisés dans une tâche (extraction, chargement, nettoyage des données). Les méta-données

[3] http://www.dbnet.ece.ntua.gr/~dwq/
[4] http://www.ifi.unizh.ch/groups/dbtg/Projects/SIRIUS/sirius.html

contiennent des informations systèmes sur les sources opérationnelles, l'évolution des composants, les règles de transformation et d'extraction des données source.

3.4.3 Squirrel

Squirrel [Zhou, et al 1995a, 1995b, 1996] [Hull, Zhou 1996] est un système de l'université du Colorado qui propose un cadre pour l'intégration de données basé sur la notion de médiateur d'intégration. Les médiateurs d'intégration sont des modules actifs qui supportent des vues intégrées au travers de multiples bases de données. Un médiateur Squirrel consiste en un processeur de requêtes, un processeur de mise à jour incrémentale, un processeur d'attribut virtuel et un système de stockage des vues matérialisées. Dans un médiateur, une vue peut être complètement matérialisée, partiellement matérialisée ou complètement virtuelle.

Les requêtes qui sont adressées au médiateur sont traitées par le processeur de requêtes en utilisant une vue matérialisée ou en ayant recours aux bases de données sources (si l'information requise n'est pas stockée au niveau du médiateur). Le processeur de mise à jour maintient les vues matérialisées de façon incrémentale.

L'architecture d'un médiateur dans Squirrel comporte trois composants : un ensemble de règles actives, un modèle d'exécution pour de telles règles et un plan de décomposition de vue. La notion de plan de décomposition de vue est analogue à celle de plan de décomposition de requête dans les optimisations de requêtes. Plus précisément, le plan de décomposition de vue spécifie les classes que le médiateur maintient et fournit la structure de base pour supporter la maintenance incrémentale. Une extension du cadre Squirrel dans lequel à la fois les vues partiellement matérialisées et les vues virtuellement intégrées sont possibles est présenté dans [Hull, Zhou 1996] [Zhou, et al 1996].

3.4.4 TSIMMIS[5]

TSIMMIS (*The Stanford-IBM Manager of Multiple Information Sources*) [Chawathe, et al 1994] [Papakonstantinou, et al 1995] [Garcia-Molina, et al 1995] est un projet visant à fournir des outils pour un accès intégré à des entrepôts d'information. Chaque source d'information est équipée d'un traducteur qui encapsule la source, convertissant les objets sous-jacents dans un modèle de données commun. Dans TSIMMIS, un modèle de données orienté-objet simple, appelé "*Object Exchange Model*" (OEM) est utilisé. Au dessus des traducteurs, TSIMMIS comporte un autre type de composants appelés médiateurs. Chaque médiateur obtient les informations d'un ou de plusieurs traducteurs ou d'autres médiateurs, affinent cette information par intégration et résolution de conflits entre les différents morceaux d'information issus des différentes sources, et fournit l'information résultant à l'utilisateur ou à d'autres médiateurs. Au niveau conceptuel, les médiateurs peuvent être considérés comme des vues sur les données d'une ou plusieurs sources qui sont soigneusement intégrées et traitées. Le médiateur est défini en terme de langage logique, appelé MSL, qui est essentiellement Datalog, étendu pour supporter les objets OEM. Les médiateurs fournissent des vues virtuelles puisqu'ils ne stockent pas les données localement. Cependant, il est possible qu'un certain médiateur matérialise la vue qu'il fournit.

[5] http://www-db.stanford.edu/tsimmis/

Quand un utilisateur adresse une requête au système, un médiateur spécifique est désigné. Ce médiateur décompose la requête et propage les sous requêtes obtenues aux niveaux inférieurs. Les réponses fournies par les différents niveaux sont alors intégrées pour constituer la réponse à la requête initiale. Le langage de requêtes de TSIMMIS est une version de SQL adaptée pour supporter les objets OEM.

L'ajout d'une nouvelle source d'information à TSIMMIS requiert la construction d'un traducteur pour la source et la modification de tous les médiateurs qui utiliseront cette nouvelle source. Les recherches dans le projet TSIMMIS ont permis de mettre au point des techniques de génération automatique de traducteurs et de médiateurs.

3.4.5 WHIPS[6]

WHIPS (*WareHouse Information Prototype at Stanford*) [Widom 1995] [Hammer, et al 1995] [Wiener, et al 1996] est un système de gestion d'entrepôts de données utilisé comme banc d'essai. Le but de ce projet est de développer des algorithmes pour collecter, intégrer et maintenir des informations émanant de sources hétérogènes, distribuées et autonomes. L'architecture du prototype WHIPS consiste en un ensemble de modules indépendants implantés comme des objets CORBA. Le composant central du système est l'intégrateur, auquel tous les autres modules sont reliés. Différents modèles de données peuvent être utilisés à la fois pour chacune des sources et pour les données de l'entrepôt. Le modèle relationnel est utilisé comme modèle unificateur : pour chaque source et l'entrepôt, les données correspondantes sont converties dans le modèle relationnel par un traducteur spécifique.

Les vues sont définies par l'administrateur système au moyen d'un sous-ensemble SQL qui inclut la sélection, la projection et la jointure. La définition d'une vue est transmise au module spécifieur de vues qui la compile dans une structure interne, appelée arbre-vue qui comporte des informations issues d'un gestionnaire de méta-données. Pour l'initialisation de l'entrepôt, l'intégrateur crée un gestionnaire de vues pour chaque vue dans l'entrepôt. Le gestionnaire de vues envoie les requêtes issues de la définition d'une vue au processeur de requêtes. Le processeur de requêtes contacte les traducteurs de la source d'information adéquate pour obtenir les résultats, les assemble et passe la réponse au gestionnaire de vue. Le gestionnaire de vue envoie la réponse au traducteur de l'entrepôt pour initialiser la vue.

Les moniteurs détectent les changements qui se produisent au niveau des sources. Il existe un moniteur pour chaque source. Les mises à jour sont transmises à l'intégrateur, qui à son tour les transmet aux gestionnaires de vues qui sont concernés par les modifications. Pour déterminer les modifications concernant la vue, un gestionnaire de vues peut générer des requêtes, qui sont passées au processeur de requêtes et évaluées au moment de l'initialisation de la vue. Les résultats des requêtes sont combinés avec les mises à jour et les modifications qui en résultent et sont envoyées au traducteur de l'entrepôt qui les applique à la vue de l'entrepôt.

3.4.6 Bilan

Ces projets sont essentiellement centrés sur des problématiques liées à l'extraction et à la maintenance des données de l'entrepôt (à l'exception de DWQ qui traite de la qualité des

[6] http://www-db.stanford.edu/warehousing/

entrepôts de données). Le modèle le plus souvent utilisé est le modèle relationnel associé au langage SQL pour définir les vues de l'entrepôt.

Les aspects plus conceptuels comme la modélisation des entrepôts et des magasins de données sont peu traités. En particulier, la modélisation du temps dans les entrepôts est peu abordée ; [Yang, Widom 1998, 2000], dans le cadre du projet WHIPS, proposent des algorithmes de maintenance pour des vues temporelles mais le modèle proposé (modèle relationnel temporel avec ajout des dates de début et de fin aux n-uplets) et le langage de définition des vues (sous-ensemble de TSQL2) sont limités. En particulier, il n'y a pas de mécanisme d'archivage des données anciennes qui permet de réduire le volume des données temporelles.

3.5 Offre commerciale Oracle Express

Plusieurs constructeurs (Hyperion Solutions, Cognos, Pilot Software, Microsoft, Oracle) proposent une offre décisionnelle. Il s'agit d'outils OLAP visant à améliorer les prises de décision au sein des entreprises. En particulier, *Oracle* propose une solution décisionnelle, appelée *Oracle Express* comprenant les modules suivants :

- un système de gestion de bases de données multidimensionnelles (MOLAP) *Oracle Express Server*,
- un outil administrateur *Oracle Express Administrator* dédié à la construction graphique de bases de données multidimensionnelles,
- un outil utilisateur *Oracle Express Analyser* permettant l'interrogation multidimensionnelle et graphique des bases de données générées.

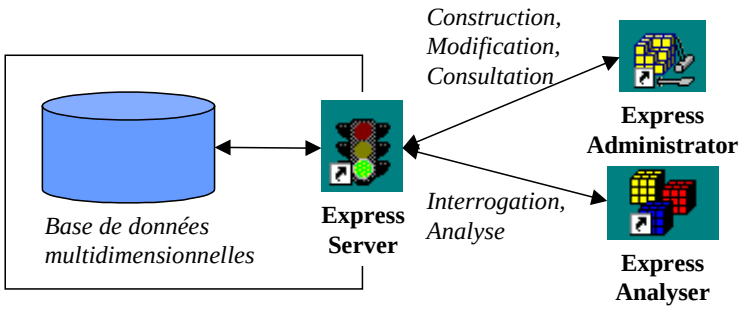


Figure 10 : Architecture d'Oracle Express.

Nous présentons dans ce qui suit le fonctionnement et l'utilisation d'*Oracle Express Administrator* et d'*Oracle Express Analyser*. Il s'agit d'opérer en trois étapes :

- la création du schéma de la base *Express*,
- l'instanciation de ce schéma par extraction des données source,
- l'exploitation de la base *Express*.

Les deux premières étapes sont réalisées au travers d'*Oracle Express Administrator* tandis que la troisième étape s'effectue avec *Oracle Express Analyser*.

3.5.1 Création du schéma de la base Express

Le module *Oracle Express Administrator* permet de construire une base de données multidimensionnelle, comprenant différents types d'éléments : les dimensions, les variables, les formules, les relations,…

> *EXEMPLE :* Pour illustrer nos propos, nous utilisons le schéma de base, en forme de flocon, présenté dans la Figure 11. Il correspond à l'analyse des ventes (quantité livrée, remise effectuée, taxes) en fonction de divers paramètres (temps, type de produits, type de clients et de fournisseurs) dans une pharmacie.

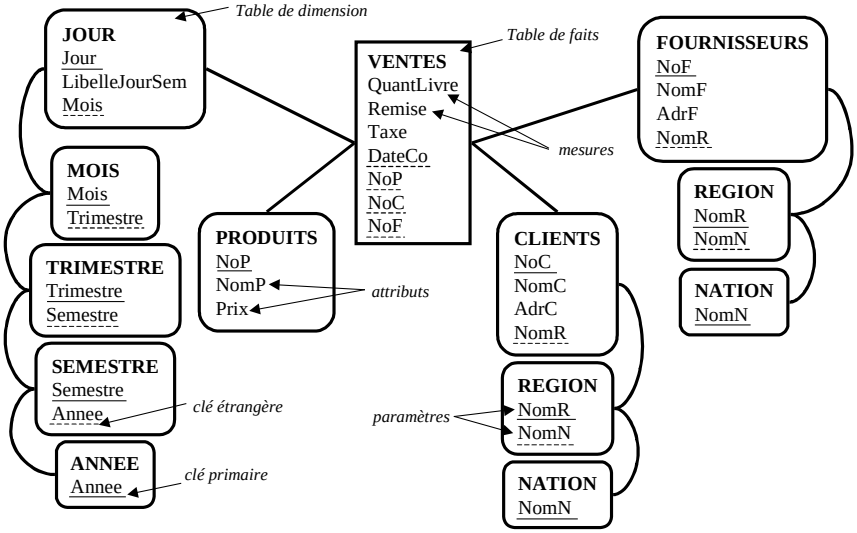


Figure 11 : Exemple d'un schéma en flocon d'analyse des ventes.

3.5.1.1 Dimensions

Une dimension est créée :

- pour chaque table de dimension "la plus fine" (table de dimension directement liée à la table de faits) dans le schéma en flocon,
- pour chaque attribut (non paramètre) des tables de dimension qui n'intervient pas en tant que niveau de détail.

> *EXEMPLE :* Pour implanter le schéma de la Figure 11, il convient de créer une dimension correspondant à chaque table de dimension ("Produits", "Jour", "Fournisseurs" et "Clients"), ainsi qu'une dimension pour chaque attribut des différentes tables de dimension ("LibelleJourSem", "NomP", "Prix", "NomC", "AdrC", "NomF", "AdrF").

3.5.1.2 Relations

Les relations permettent d'établir la liaison entre une table de dimension et ses attributs respectifs. Une relation est construite entre la dimension (représentant une table de dimension dans le schéma en étoile) et chaque attribut de cette dimension.

> *EXEMPLE :* Dans l'exemple, on crée une relation reliant les dimensions ("Temps", "Produits", "Clients", "Fournisseurs") à chacun de leurs attributs respectifs ("LibelleJourSem", "NomP", "Prix", "NomC", "AdrC", "NomF", "AdrF").

3.5.1.3 Variables, Formules

Les variables correspondent aux mesures de l'analyse contenues dans la table de fait (attributs non clé de la table de fait).

> *EXEMPLE :* Pour l'implantation de l'exemple, nous devons créer les trois variables "QuantLivre", "Remise", "Taxe".

Les formules sont des expressions de calcul élaborées à partir des variables et des dimensions. Le résultat d'une formule est calculé dynamiquement au moment de l'interrogation.

> *EXEMPLE :* Dans l'exemple, nous pouvons ajouter la formule "montant" correspondant à l'expression de calcul suivante : QuantLivre*Prix.

La figure suivante décrit chacun des éléments de la base *Express* en utilisant les menus contextuels et en définissant leurs caractéristiques.

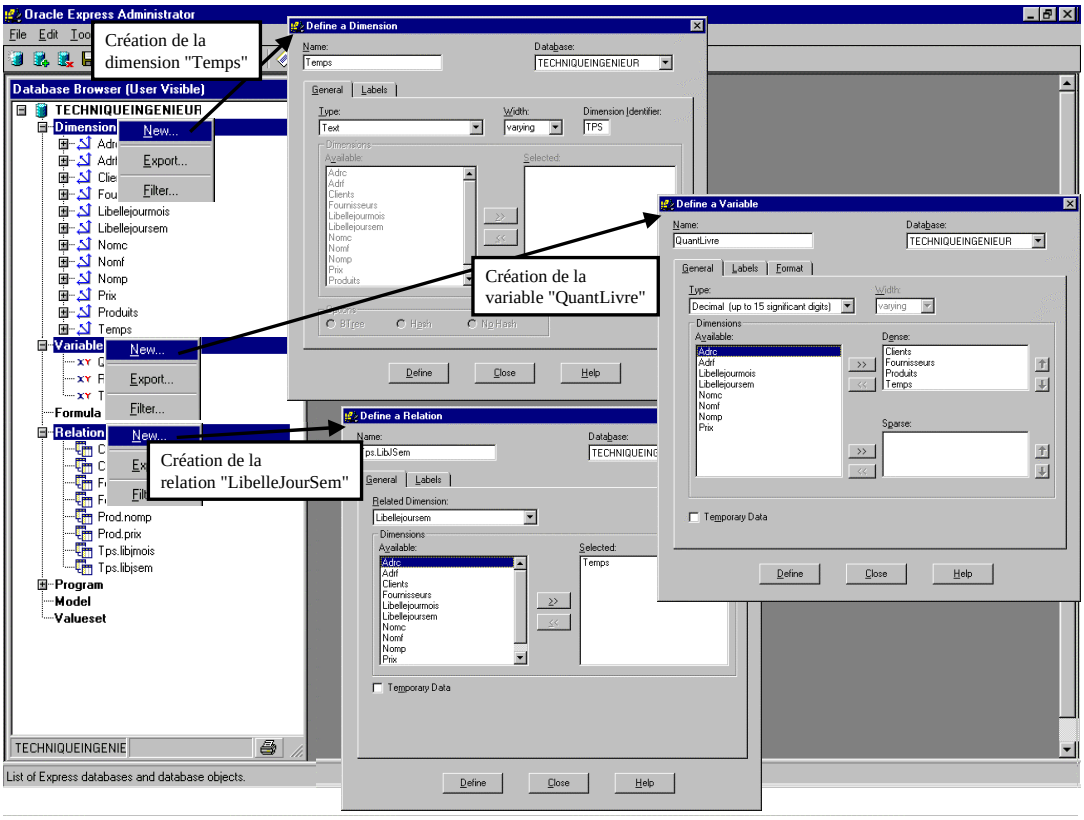


Figure 12 : Représentation dans Express du schéma en flocon.

3.5.2 Importation des données de la source vers la base Express

L'importation des données source consiste en la valorisation des différents éléments de la base *Express*. Elle nécessite une étape préliminaire qui extrait d'une base source, via une requête SQL, toutes les données nécessaires (équivalent de la relation universelle). Le résultat est stocké dans un fichier texte intermédiaire. L'importation comprend plusieurs étapes :

- importation des valeurs contenues dans les dimensions,
- création des hiérarchies entre les valeurs des dimensions,
- création des relations entre les dimensions et leurs attributs,

- importation des variables (ou mesures) et calcul d'agrégations

3.5.2.1 Importation des valeurs des dimensions

Pour chaque dimension, on importe les valeurs de ses paramètres, appelées les positions.

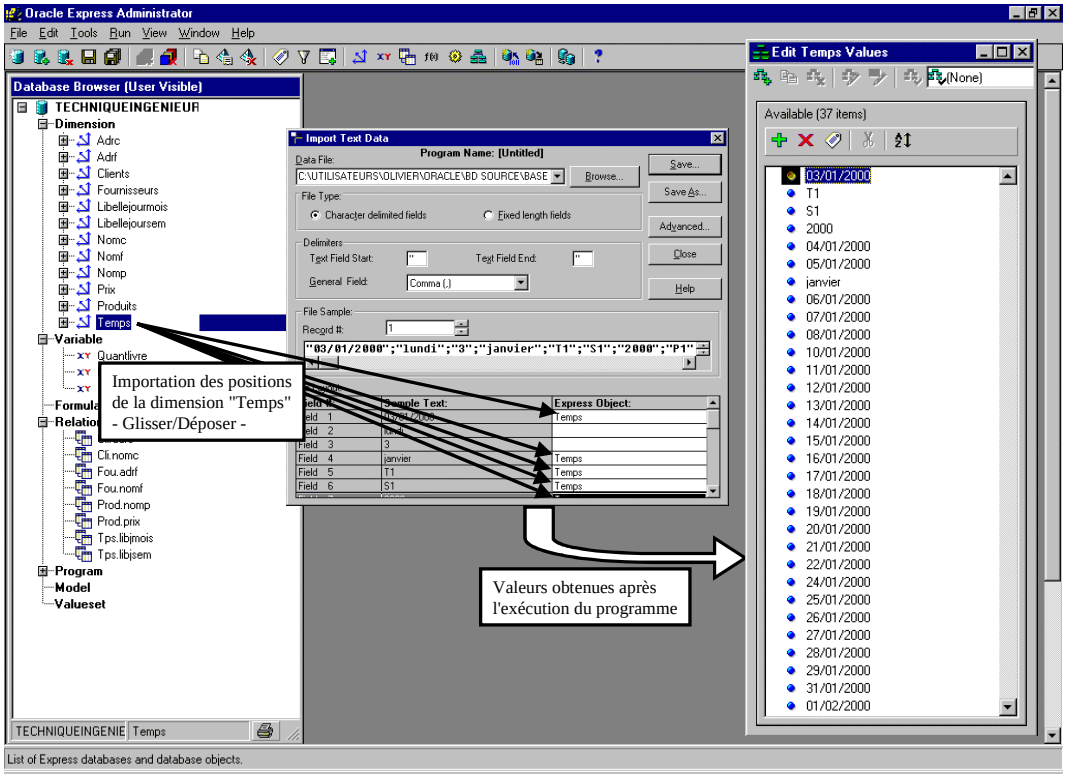


Figure 13 : Exemple d'importation des positions de la dimension temps.

Il est important de remarquer que les données temporelles doivent être présentes dans la source de données. Autrement dit, *Oracle Express Server* ne permet pas d'historiser les données de l'entrepôt (c'est à dire de conserver l'évolution des données au cours du temps).

3.5.2.2 Création des hiérarchies des dimensions

Pour chaque dimension, on construit une hiérarchie entre ses paramètres. Sur l'exemple de la dimension "Temps", l'an 2000 comprend le semestre S1,…

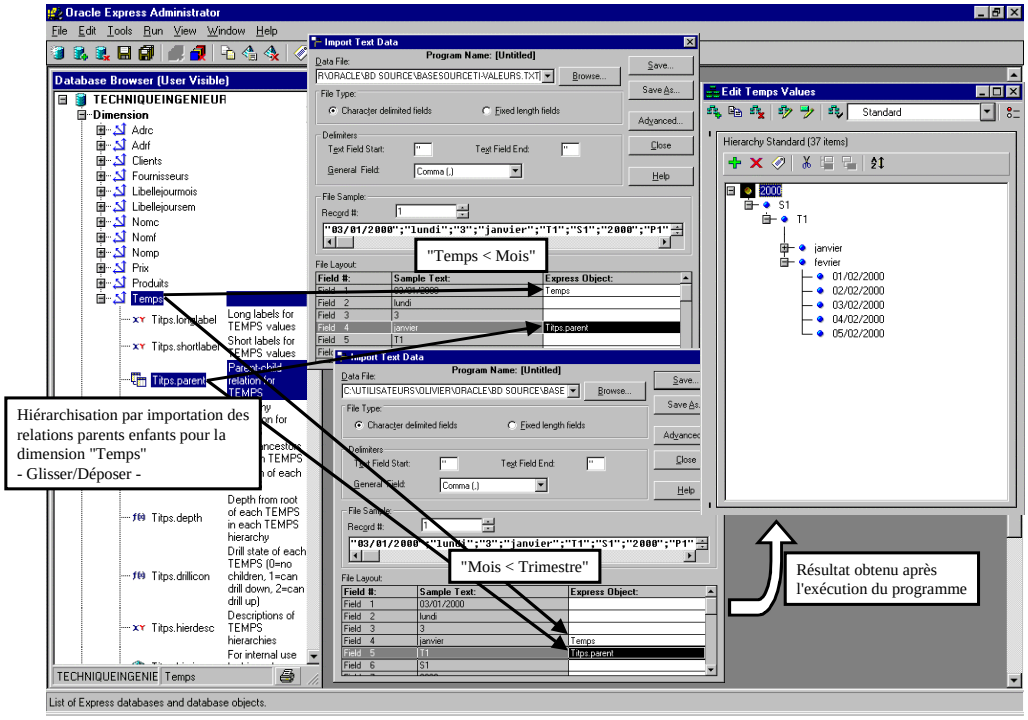


Figure 14 : Exemple de hiérarchisation de la dimension temps.

3.5.2.3 Création des relations entre les dimensions et leurs attributs

Cette importation consiste à valoriser les relations.

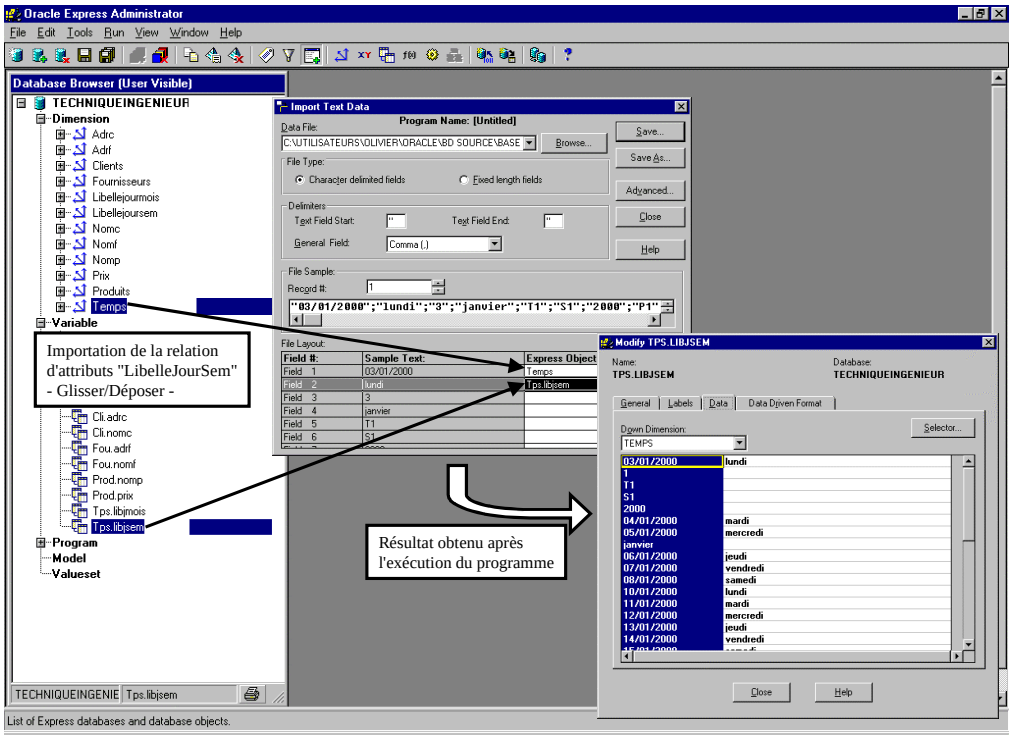


Figure 15 : Exemple d'importation de la relation de l'attribut "LibelleJourSem".

3.5.2.4 Importation des variables

Pour chaque variable, on importe les valeurs en précisant les dimensions associées.

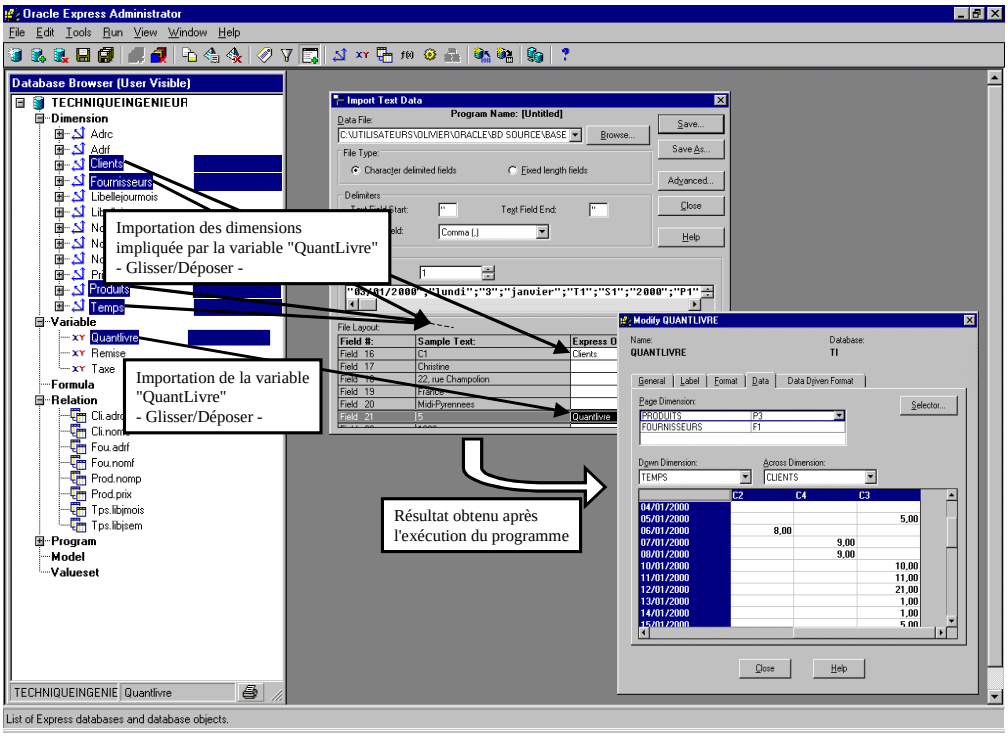


Figure 16 : Exemple d'importation de la variable "QuantLivre".

Enfin, via le menu "Tools", l'administrateur lance un processus automatique qui calcule les agrégats des différents niveaux de détail pour améliorer les performances lors de l'interrogation et de l'analyse de la base *Express*.

3.5.3 Manipulation de la base Express

Le module *Oracle Express Analyser* permet de manipuler une base Express. Les données sont représentées sous forme d'hypercubes où les arêtes sont constituées par les dimensions et les cases contiennent les variables. Un hypercube possède autant d'arêtes qu'il y a de dimensions ; l'outil *Express Analyser* permet de visualiser le plan d'affichage de deux dimensions. Les autres plans sont visualisés en changeant les dimensions.

Par exemple, on peut visualiser d'hypercube correspondant à la variable "QuantLivre", en créant un "*briefing*" contenant les plans suivant les dimensions "Jours", "Produits", "Clients" et "Fournisseurs". Sur la figure suivante, on affiche le plan des dimensions "Jours" (en ligne) et "Clients" (en colonne). Puis on visualise la tranche correspondant au fournisseur "F1" et au produit "P3".

Express Analyser manipule donc des tables. Plusieurs opérations multidimensionnelles sont disponibles : forage vers le bas et vers le haut ("drill down" / "roll up"), rotation, c'est à dire inversion de lignes/colonnes/plans ("rotate"), sélection de positions de dimensions ("select"), emboîtement ("nest" / "unest").

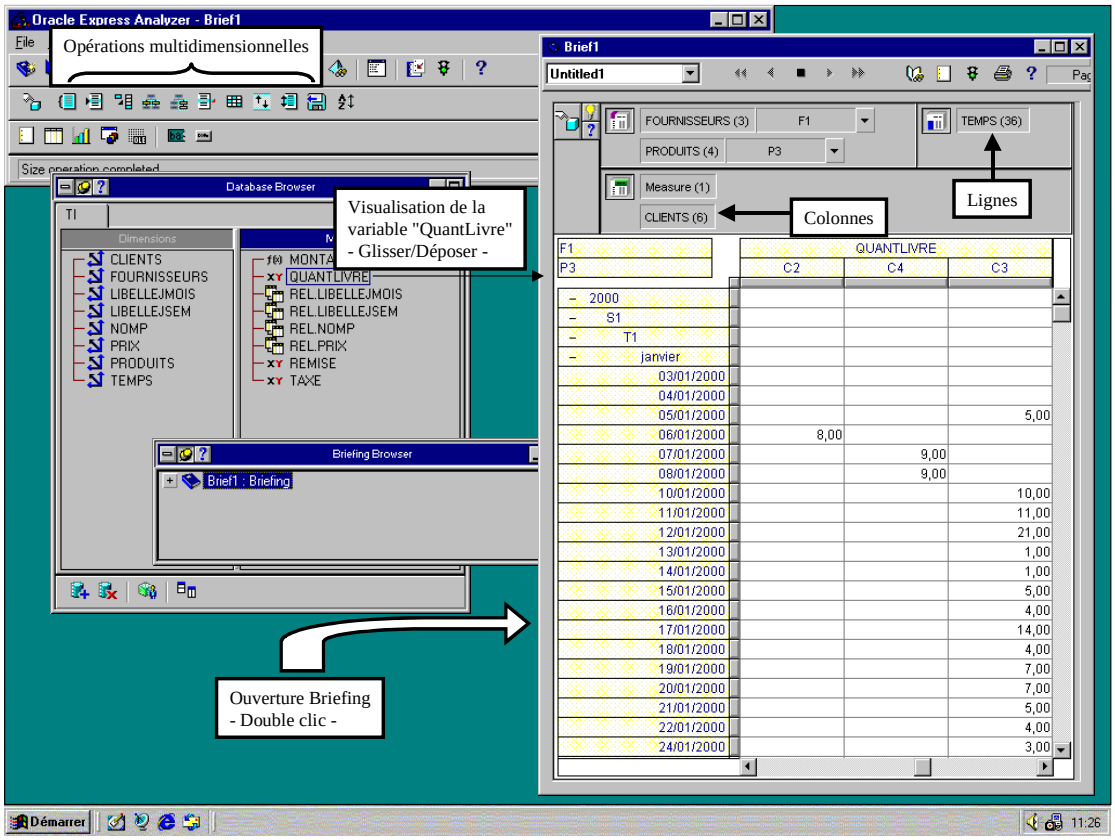


Figure 17 : Exemple de "*briefing*" pour la variable "QuantLivre".

3.5.4 Bilan

Nous avons présenté l'outil *Oracle Express*. Il permet de construire un entrepôt (ou magasin) à partir de données issues des bases de production et il donne la possibilité de manipuler cette base multidimensionnelle en visualisant l'information sous forme de tableau correspondant à un plan dans l'hypercube.

Mais cet outil, comme les autres offres commerciales actuelles (*cf.* annexe B), présente des limites.

- La définition du schéma est fastidieuse et demande une expertise complète de la part de l'administrateur. Ce dernier doit connaître parfaitement l'information contenue dans les sources de données pour effectuer manuellement toutes les correspondances avec les données de l'entrepôt.
- L'outil *Express* ne distingue pas l'entrepôt des magasins de données (cf. section 2.3). Cette approche ne permet donc pas de séparer les problématiques spécifiques à l'entrepôt et aux magasins. La maintenance de l'entrepôt n'est pas réalisée et il est nécessaire d'effectuer une extraction complète à chaque fois que l'entrepôt doit être rafraîchi.
- L'historisation des données au niveau de l'entrepôt n'est pas réalisée. Ceci constitue une limite importante car la conservation des évolutions des données est une caractéristique essentielle pour supporter les prises de décision [Inmon 1994] [Yang, Widom 1998, 2000] [Mendelzon, Vaisman 2000].

3.6 Synthèse

Les entrepôts (et les magasins) de données sont un thème de recherche important [Widom 1995] [Chaudhuri, Dayal 1997]. Nous avons identifié deux axes de recherche majeurs :

- l'approche des vues matérialisées,
- la modélisation multidimensionnelle.

Les travaux actuels ne distinguent pas l'entrepôt de données lui-même des magasins de données. Or ces deux espaces de stockage ont des objectifs très différents (*cf.* section 2.3) et doivent être considérés de manière indépendante.

Ainsi, des limites importantes peuvent être identifiées.

- Les travaux actuels concernant la modélisation proposent une organisation multidimensionnelle des données. Or ce type de modélisation est bien adapté aux objectifs des magasins de données (supporter efficacement les analyses) mais se révèle inadapté à ceux de l'entrepôt (gestion efficace des données et surtout des données historisées, pérennité de ces données dans le temps). En effet, l'approche multidimensionnelle induit d'importantes redondances d'information liées à la dénormalisation [Codd 1993] [Kimball 1996].
- Les études abordant l'aspect (essentiel) des données temporelles dans les entrepôts restent peu nombreuses et n'offrent pas la puissance des modèles temporels tels qu'ils sont définis dans les bases de données classiques. A notre connaissance, aucune étude n'est réalisée concernant l'archivage des données dans le domaine des entrepôts de données. De plus dans les modèles multidimensionnels, la dimension temporelle est traitée de manière identique aux autres dimensions des magasins. Ceci permet d'appliquer au temps les opérations multidimensionnelles mais ne permet pas des requêtes temporelles complexes.

4 ETAT DE L'ART SUR L'EVOLUTION DES DONNEES

Cette section présente un état de l'art concernant les différentes propositions actuelles dans le domaine des bases de données temporelles et à version qui gèrent les évolutions des données. Cette étude vise à identifier les fondements essentiels à intégrer dans les entrepôts de données tant au niveau des modèles que des langages de manipulation.

Nous présentons les travaux effectués dans le domaine des bases de données temporelles et dans le domaine des bases de données intégrant les versions. Nous étudions plus particulièrement les travaux dans les bases de données objet, tout en intégrant certains travaux significatifs du relationnel.

4.1 Bases de données temporelles

Les bases de données temporelles offrent les concepts et les fonctionnalités permettant aux applications de dater les informations et de gérer l'histoire de leurs évolutions, contrairement aux bases de données traditionnelles qui offrent une vision instantanée des données (définies par la dernière mise à jour de la base).

4.1.1 Modèles temporels

De nombreux travaux ont été réalisés pour intégrer le temps dans les bases de données[7] [Tsotras, Kumar 1996] [Wu, et al 1998]. Ces travaux peuvent être divisés en deux approches, selon qu'ils datent les informations au niveau des objets (ou n-uplets) et au niveau des valeurs d'attributs :

1. La première approche consiste à associer un intervalle de temps à l'objet. A chaque modification de la valeur d'un des attributs, un nouvel état est décrit. Ce nouvel état contient une valeur pour chacun des attributs. D'un état à l'autre, les attributs n'ayant pas été modifiés ont la même valeur. Cette approche entraîne des problèmes de redondance car une même information est conservée plusieurs fois ; pour résoudre se problème, le concept de stockage du différentiel (delta) a été proposé.
2. La deuxième approche consiste à associer un intervalle de temps aux attributs. Chaque changement de la valeur d'un attribut est conservé. La valeur qu'il possédait précédemment est associée à un intervalle de temps borné et la nouvelle valeur à un intervalle de temps non borné. Les différentes valeurs prises par un attribut sont conservées indépendamment des changements de valeurs des autres attributs.

Certains travaux comme OODAPLEX [Dayal, Wuu 1992] [Wuu, Dayal 1993] intègrent les deux possibilités en permettant de dater les données au niveau des objets ou des attributs.

Dans le Tableau 7 nous étudions les principaux travaux de recherche dans le domaine des bases de données temporelles. Notre étude comparative est effectuée selon différents critères.

- La première colonne précise le modèle de données (objet, E/R ou relationnel).
- La seconde colonne indique le niveau temporel. Les informations sont datées soit au niveau des objets (1) soit au niveau des attributs (2).
- La troisième colonne spécifie la densité du temps (discrète lorsque le temps est considéré comme non continu ou dense lorsqu'il est vu comme une droite continue).
- La quatrième colonne liste les types temporels de base supportés par le modèle. Les types temporels de base considérés sont la durée, l'instant et l'intervalle. Certains travaux tels que TEMPOS offrent des types temporels plus complexes (chroniques, historiques…) appelés éléments temporels.
- Les cinquième et sixième colonnes donnent la sémantique du temps utilisé pouvant être
  - un temps de validité (temps durant lequel l'information est vraie dans le monde réel),
  - un temps de transaction (temps reflétant l'instant auquel l'information a été enregistrée dans la base de données).

[7] http://isse.gmu.edu/~csis/tdb/bib97/bib97.html *propose une bibliographie, classée en douze catégories, des différents travaux de recherche entre 1995 - 1997 sur les bases de données temporelles.*

	Modèle de données	Niveau temporel	Densité du temps	Types temporels	Temps validité	Temps transaction
ERT [Theodoulidis, et al 1994]	E/R	1 - 2	Discret	Durée, Instant, Intervalle	Oui	Non
OODAPLEX [Dayal, Wuu 1992] [Wuu, Dayal 1993]	Objet	1 - 2	Définie par utilisateur	Instant, Ensemble d'instants	Oui	Oui
OSAM*T [Su, Chen 1991]	Objet	1	Discret	Instant, Intervalle	Oui	Non
T_ODMG [Bertino, et al. 1998a]	Objet	1 - 2	Discret	Durée, Instant, Intervalle	Oui	Oui
TAU [Kakoudakis, Theodoulidis 1996]	Objet	1 - 2	Discret	Durée, Instant, Intervalle	Oui	Oui
T-Chimera [Bertino, et al 1996]	Objet	2	Discret	Instant, Intervalle	Oui	Oui
TEER [Elmashri, Wuu 1990] [Kouramajian, Gertz 1995]	E/R étendu	1 - 2	Discret	Non	Oui	Non
TEMPOS [8] [Canavaggio 1997]	Objet	2	Discret	Durée, Instant, Intervalle	Oui	Oui
TIGUKAT [Goralwalla, Özsu 1993] [Özsu, et al 1993, 1995]	Objet	1 - 2	Discret, Dense	Durée, Instant, Intervalle	Oui	Non
TMQL [Käfer, Schöning 1992a]	Objet	1	Discret	Durée, Instant, Intervalle	Oui	Oui
TOOBIS [9] [Toobis 1998a, 1998b, 1998c]	Objet	1 - 2	Discret	Durée, Instant, Intervalle	Oui	Oui
TOODM [Rose, Segev 1991, 1993a, 1993b]	Objet	2	Discret	Instant, Intervalle	Oui	Oui
TSQL2 [Snodgrass 1995]	Relationnel	1	Discret, Dense	Durée, Instant, Intervalle	Oui	Oui

Tableau 7 : Comparaison de modèles temporels.

Parmi ces travaux, plusieurs propositions visent à étendre le standard pour les bases de données objet proposé par l'ODMG : TOOBIS [Toobis 1998a, 1998b, 1998c], TEMPOS [Canavaggio 1997], T_ODMG [Bertino, et al. 1998a], TAU [Kakoudakis, Theodoulidis 1996]. Cependant, aucune extension temporelle n'apparaît comme le standard des bases de données temporelles objet (contrairement aux bases de données relationnelles où TSQL2 [Snodgrass 1995] et SQL3 [Snodgrass, et al 1998] constituent des standards).

[8] *Le modèle TEMPOS s'inscrit dans le projet STORM (http://www-lsr.imag.fr/storm.html) visant à définir des nouvelles architectures, fonctions, langages pour les futurs Systèmes de Gestion de Bases de Données.*

[9] *Le projet TOOBIS est un projet Européen Esprit IV réalisé par l'Université d'Athènes en Grèce ; http://www.mm.di.uoa.gr/~toobis/.*

En outre, dans les bases de données temporelles, les évolutions des valeurs sont toutes conservées. Ceci induit l'inconvénient majeur de générer des volumes de données très importants, altérant rapidement le fonctionnement de la base. Des données temporelles doivent alors être soit supprimées de la base ou bien archivées. Or cet archivage n'est pas réalisé automatiquement et reste à la charge des administrateurs.

4.1.2 Langages temporels

Au niveau des langages, il existe de nombreuses extensions temporelles dans les bases de données objet. Nous résumons les principaux opérateurs généralement proposés.

- La **projection temporelle** est une extension directe de la projection classique. Par exemple, "*Pendant quelle période le docteur Dupond a-t-il dirigé le service des urgences de Rangueil*". Si le temps est associé aux attributs, la réponse est le temps associé à la valeur "*Dupond*" de l'attribut *Chef_Service*. Si le temps est associé aux n-uplets, la réponse est l'union des temps associés aux n-uplets dont la valeur de l'attribut *Chef_Service* est "*Dupond*".
- La **sélection** et la **restriction temporelle** permettent de retenir l'histoire de n-uplets vérifiant une certaine propriété à certains moments dans le temps. Par exemple, "*Retrouver les directeurs ayant dirigés le service d'urgence de Rangueil sur la période 1980 - 2000*".
- Les **opérateurs de comparaison** permettent de combiner les types temporels et les opérateurs d'ordonnancement permettent de retrouver le premier, le dernier ou le n-ième élément d'un historique temporel. Par exemple, les relations de Allen [Allen 1983] permettent de manipuler les intervalles de temps.
- Le **produit cartésien temporel** étend le produit cartésien selon deux options : considérer que la période de la validité des n-uplets du résultat est l'union ou l'intersection des périodes de temps des n-uplets arguments.
- Les **opérateurs ensemblistes** sont redéfinis de manière à traiter correctement les cas de chevauchement des périodes de temps.

4.1.3 Bilan

Les travaux traitant de l'extension temporelle des bases de données visent à intégrer toutes les fonctionnalités nécessaires à la prise en compte du temps. Deux approches de modélisation sont proposées ; elles permettent de conserver les évolutions soit au niveau des attributs, soit au niveau des objets. L'accroissement de la complexité des requêtes dans les bases de données temporelles est prise en compte par la définition de nouveaux langages de requêtes de haut niveau intégrant divers opérateurs temporels.

Néanmoins, les bases de données temporelles se heurtent souvent au volume des données engendré par l'historisation. Cette difficulté est d'autant plus aiguë que dans notre contexte d'entrepôt de données, les volumes de données sont encore plus importants. La seule solution mise en œuvre actuellement consiste à supprimer les historiques trop anciens. A notre connaissance, aucune étude n'est réalisée pour permettre d'archiver les données temporelles à des niveaux de détail plus important, ceci afin de conserver une trace de l'histoire des données

tout en limitant le volume engendré. Or l'archivage automatique des données est une des caractéristiques que nous souhaitons intégrer au modèle dédié aux entrepôts.

4.2 Bases de données intégrant les versions

Les bases de données intégrant les versions permettent de conserver et de manipuler les évolutions des données en conservant différentes valeurs appelées versions pour un objet.

4.2.1 Modélisation et manipulation des versions

Dans les bases de données intégrant les versions, l'évolution d'une entité du monde réel peut être traduite :

- au niveau de la valeur (ou niveau de l'objet) ; c'est le cas des systèmes Charly [Palisser 1989], OVM [Käfer, Schöning 1992b], O2 [O2tech 1995],
- au niveau du schéma (ou niveau de la classe) ; c'est le cas des systèmes Encore [Skarra, Zdonik 1986] [Shaw, Zdonik 1990], OTGen [Lerner, Habermann 1990], CloSQL [Monk, Sommerville 1993].

D'autres systèmes proposent les deux niveaux comme Orion [Chou, Kim 1986], Iris [Beech, Mahbod 1988], Avance [Björnersted, Hultén 1989], Presage [Talens, et al 1993] ainsi que les travaux effectués au sein de notre équipe et relatifs à VOHQL [Andonoff, et al 1995] [Hubert 1997] [Le Parc 1997].

Parmi tous ces travaux, deux approches se distinguent : les versions de contexte et les serveurs de versions.

Le principe des **versions de contexte** est de regrouper, au sein d'un même contexte, des versions d'entités selon une signification précise. Chaque contexte reflète, pour cette signification, un état du monde réel. La notion de **contexte** permet de se placer dans un espace de travail particulier correspondant à un état du monde réel en faisant abstraction des autres états.

Le principe des **serveurs de versions** est de décrire les évolutions au travers de versions d'objet et de classe. L'évolution de valeur est décrite par les versions d'objet tandis que les évolutions de schéma sont décrites par les versions de classe et/ou de schéma.

- Les versions au niveau objet permettent de décrire les évolutions de valeurs des entités. Une entité est alors décrite par un ensemble de versions appelées versions d'objet. Une version d'objet correspond à une valeur significative de l'entité au cours de son cycle de vie. Une entité est initialement représentée par une seule version, appelée version racine. La représentation de toute évolution de cette entité dans la base de données induit soit la modification de la version, soit la définition d'une nouvelle version, obtenue à partir d'une version précédente par dérivation. Lorsque plusieurs versions dérivent d'une même version, les versions sont dites **alternatives** ; le processus de dérivation peut donc être linéaire (sans alternative) ou non linéaire.
- Les versions de classes et de schémas permettent de décrire et de conserver l'évolution de schéma des entités du monde réel. Certains systèmes ne permettent la gestion des versions qu'au niveau des classes. D'autres systèmes utilisent les versions de schéma pour décrire l'évolution de schéma des entités. Les versions de schéma permettent de gérer l'évolution

d'un ensemble de classes. Certains systèmes ne permettent la gestion des versions qu'au niveau des classes. Trois approches différentes sont utilisées pour adapter les objets à la nouvelle spécification de la classe :

- la **conversion** consiste à mettre à jour tous les objets non compatibles avec le nouveau schéma,
- l'**émulation** consiste à simuler la sémantique de la nouvelle interface pour les objets de l'ancienne version,
- le **versionnement** consiste à associer une version à chaque version de classe.

Le Tableau 8 compare les différents systèmes de gestion de bases de données intégrant le concept de version :

- la première colonne indique par quel principe les évolutions de valeur sont conservées (version d'objet, version de contexte),
- la seconde colonne indique par quel principe les évolutions de schéma sont conservées (version de classe, version de schéma),
- la troisième colonne précise le processus de dérivation supporté (linéaire, non linéaire, avec ou sans alternatives).

	Evolution de valeur	**Evolution de schéma**	**Processus de dérivation**
Avance [Björnersted, Hultén 1989]	Version d'objet	Version de classe	Linéaire + non linéaire
Charly [Palisser 1989]	Version d'objet		Linéaire + alternatives (2 niveaux maximum)
CloSQL [Monk, Sommerville 1993]		Version de classe (conversion)	Linéaire
Encore [Skarra, Zdonik 1986]		Version de classe (émulation)	Linéaire
Iris [Beech, Mahbod 1988]	Version d'objet	Version de schéma (conversion)	Linéaire + non linéaire
O2 [O2tech 1995]	Version de contexte (configurations)		Linéaire + non linéaire
Orion [Chou, Kim 1986]	Version d'objet	Version de schéma (conversion)	Linéaire + non linéaire
OTGen [Lerner, Habermann 1990]		Version de schéma (conversion)	
OVM [Käfer, Schöning 1992b]	Version partielle d'objet		Linéaire
Presage [Talens, et al 1993]	Version d'objet	Version de classe (indépendante)	Linéaire + alternative
VBD [Cellary, Jomier 1990] [Gançarski 1994]	Version de contexte (version de BD)	Version de schéma (conversion)	Linéaire + non linéaire
VOHQL [Andonoff, et al 1995] [Le Parc 1997]	Version d'objet	Version de classe (indépendante)	Linéaire + alternative

Tableau 8 : Comparaison des systèmes gérant des versions.

Peu de travaux sur la gestion de versions se sont intéressés aux langages de manipulation. La plupart des langages proposés sont des langages déclaratifs [Chou, Kim 1986] [Käfer,

Schöning 1992b] [Gançarski 1994]. Néanmoins, des travaux réalisés dans notre équipe SIG ont proposé non seulement le langage (de manipulation des versions) déclaratif VOSQL [Hubert 1997] mais également un langage algébrique [Le Parc 1997] ainsi que le langage graphique VOHQL [Teste 1996] [Le Parc 1997]. Ces langages se proposent de manipuler des versions comme des objets, sans considérer les valeurs temporelles.

4.2.2 Bilan

Les bases de données intégrant les versions permettent de conserver les évolutions des données et des schémas. Il existe principalement deux approches : les versions de contexte (qui regroupent au sein d'un même contexte des versions d'objets selon une signification précise, permettant ainsi de se placer dans un espace de travail particulier correspondant à un état du monde réel en faisant abstraction des autres états) et les serveurs de versions (qui décrivent les évolutions au travers de versions d'objet et de classe permettant de conserver les évolutions respectivement de valeur et de schéma).

Néanmoins, l'approche des versions n'intègre pas explicitement les valeurs temporelles et les langages associés ne manipulent pas la dimension temps. Ces limites sont importantes car elle rendent la mise en œuvre d'un mécanisme automatique d'archivage des évolutions (visant à agréger les valeurs passées) impossible.

4.3 Synthèse

La prise en compte des évolutions dans les bases de données est réalisée au travers

- d'extensions temporelles qui intègrent explicitement la dimension temporelle,
- de versions qui conservent les évolutions des objets.

Les bases de données temporelles offrent des abstractions évoluées pour modéliser le temps et manipuler les données temporelles. Néanmoins, ces travaux se heurtent au problème du volume des données. Les mécanismes d'archivage nécessaires ne sont pas automatisés et restent à l'initiative des administrateurs.

Les bases de données à version permettent de conserver l'évolution des données et de leur schéma. Cependant cette approche n'offre pas la puissance des bases de données temporelles en terme d'intégration et de manipulation directe du temps. Cette limite rend l'archivage des données délicate et complexe.

5 NOTRE PROPOSITION

5.1 Contexte des travaux

Les systèmes décisionnels sont nés d’un besoin des entreprises qui n'est pas offert par les systèmes de gestion de bases de données traditionnels ; il s'agit d'un support efficace et performant pour l'aide à la prise de décision.

Les systèmes décisionnels comportent deux types d'espaces de stockage que sont les entrepôts de données et les magasins de données.

- Un entrepôt de données regroupe dans un format homogène des données utiles pour l'aide à la décision provenant de sources internes (bases de production) ou externes (bases de partenaires, Internet…) à une entreprise.
- Un magasin de données est un extrait d'informations, orienté sujet, provenant de l'entrepôt et organisé de manière adéquate pour y appliquer des analyses rapides à des fins de prise de décision.

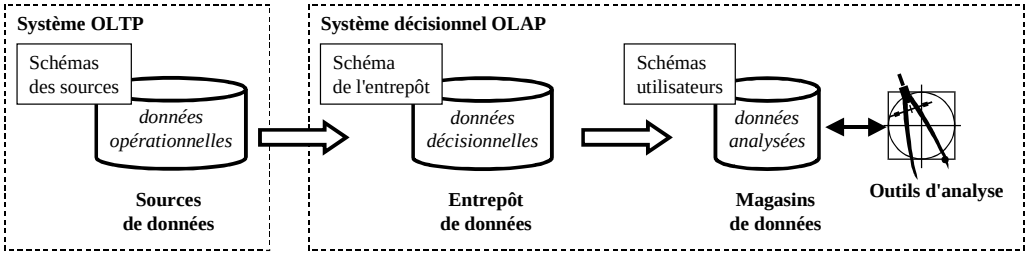


Figure 18 : Du système OLTP au système OLAP.

5.2 Cadres d'applications

La présente étude, effectuée au sein de l'équipe SIG (Systèmes d'Informations Généralisées), s'inscrit dans deux projets.

5.2.1 CTI-Sud[10]

L'Assurance Maladie collecte un nombre très important de données sur les patients, les professionnels de santé, les établissements hospitaliers… Ces données peuvent servir de base à un système décisionnel visant à supporter les contrôles, les analyses épidémiologiques (ou autres), le suivi de l'activité des médecins et des centres hospitaliers…

Or à l'heure actuelle, l'exploitation de ces données à des fins de prise de décision est réalisée au travers d'un SGBD relationnel, appelé la base SIAM, inadapté à de telles activités. Les réponses ne satisfont pas les experts de l'Assurance Maladie (tels que les médecins conseils). Les raisons principales étant que :

- la base SIAM n'est pas conçue pour supporter efficacement les processus de décision,
- elle est peu évolutive,
- elle est dédiée à de multiples applications,
- elle stocke toute l'information, même les données inutiles, incomplètes ou inexactes (l'Assurance Maladie estime le volume de ces données à 15% du volume total).

Par ailleurs, compte tenu de l'importance du volume des informations collectées, l'Assurance Maladie ne peut conserver un historique informatisé des données que sur une période de temps variant de quelques mois à deux ans maximum ; cette perte d'information temporelle rend certaines études, comme par exemple les études épidémiologiques difficiles, voire impossibles (car certaines d'entre elles nécessitent des historiques sur de nombreuses années).

[10] *La présente étude a été partiellement financée par le **CTI-Sud** (Centre de Traitement Informatique des régions Midi-Pyrénées et Languedoc-Roussillon) de l'Assurance Maladie.*

Dans le cadre d'une convention entre l'Université Paul Sabatier et le CTI-Sud, nous avons mené une expertise sur l'élaboration d'un entrepôt de données médicales évolutives. Plus précisément, nous avons proposé une solution visant à construire un système décisionnel pour l'Assurance Maladie, intégrant un entrepôt de données historisées et archivées. Cet entrepôt est alimenté par les bases opérationnelles de l'Assurance Maladie. Il collecte uniquement les données pertinentes pour les décideurs. L'évolution des données est conservée et un mécanisme automatique d'archivage est proposé pour limiter le volume de l'information.

5.2.2 REANIMATIC[11]

A l'heure actuelle, aucun outil ne permet d'aider le clinicien à prédire la survenue de complications nosocomiales ou d'aggravations des patients en réanimation. Pourtant de nombreuses données sont collectées chaque jour au chevet du patient dans les services de réanimation.

Le but du projet REANIMATIC est d'élaborer un système automatisé évolutif individuel de veille des patients de réanimation, permettant d'estimer la probabilité de survie et les probabilités de survenue de complications, dans le but d'une amélioration de la qualité des soins et du devenir des patients en réanimation. Ce système doit permettre d'une part une amélioration du pronostic des patients en tenant compte de leur évolution pendant leur séjour en réanimation, et d'autre part l'identification et la détermination des facteurs de risque ainsi que le pronostic des infections nosocomiales et des événements iatrogènes survenant pendant le séjour en réanimation, en tenant compte du moment auquel ils surviennent, et de la gravité des patients au moment de leur survenue.

Les travaux de recherche exposés dans cette thèse sont issus de notre participation à ce projet. REANIMATIC rassemble deux pôles d'expertises : l'association OUTCOME REA qui apporte l'expertise en réanimation polyvalente et des équipes du groupe EVOLUTION[12] dont l'équipe SIG, qui apportent leur expertise en conception d'entrepôts de données.

5.3 Objectif

Ces deux projets se situent dans un contexte médical et visent un même objectif : élaborer un système d'information médicale dédié aux applications décisionnelles. Ce système décisionnel se base sur l'approche des entrepôts qui collectent les données pertinentes pour les décideurs. Ces données médicales doivent être historisées et archivées afin de pouvoir maintenir l'entrepôt sur une période de temps importante.

L'objectif d'un tel entrepôt est de servir de support aux processus de décision, au travers de magasins spécialisés dans des tâches particulières.

---

[11] *Le projet **REANIMATIC**, en collaboration avec l'association de médecins OUTCOME-REA, vise à concevoir et à développer un entrepôt de données médicales et évolutives (collectées à partir des bases opérationnelles des services de réanimation) afin d'améliorer la qualité des soins et le devenir des patients dans les services de réanimation des hôpitaux français.*

[12] *Le groupe **EVOLUTION** (http://www.prism.uvsq.fr/dataware/coop/evolution.html) fédère plusieurs équipes de recherche françaises (dont l'équipe SIG, fondatrice d'EVOLUTION avec le PRISM) dans le domaine des entrepôts de données. Ce groupe se positionne dans le domaine de la conception des systèmes d'informations et propose le développement de méthodologies et d'outils de type CASE, pour l'aide à la conception et l'évolution des entrepôts de données.*

- Dans le cadre de l'Assurance Maladie, les tâches effectuées sont le contrôle de l'activité des professionnels de santé, le contrôle du comportement consommateur des patients, les études épidémiologiques (basée sur des données anonimisées),…
- Dans le cadre de REANIMATIC, les tâches effectuées telles que des simulations ou des études statistiques sont réalisées sur les données médicales collectées afin de prédire la survenue de complications nosocomiales ou d'aggravations des patients en réanimation.

5.4 Insuffisances des approches actuelles

Les travaux relatifs aux entrepôts de données abordent principalement deux problématiques.

- La première traite essentiellement de l'organisation des données. Cette organisation dite multidimensionnelle vise à supporter efficacement les analyses OLAP en offrant une vision des données adaptées et les temps de réponse sont accélérés en calculant de nombreux préagrégats.
- La deuxième étudie principalement la sélection et la maintenance incrémentale des vues matérialisées afin de collecter les données utiles aux décideurs, de les stocker dans l'entrepôt et de les maintenir cohérentes avec les données source. Ces travaux se focalisent sur des aspects physiques ou logiques (vues, index,…).

Cependant, ces travaux ne sont pas suffisants. Des limites subsistent dans l'élaboration d'un système décisionnel basé sur les entrepôts et en particulier la problématique de modélisation des données dans les systèmes décisionnels est peu traitée.

- Les travaux actuels se basent sur des modèles relationnels ou multidimensionnels qui intègrent ni des structures complexes ni une sémantique riche. Ceci oblige les concepteurs à un effort d'abstraction important afin de représenter le monde réel dans l'entrepôt. En outre, le manque de méthode ou de démarche aidant les concepteurs dans cette tâche rend cette activité difficile et fastidieuse.
- L'approche multidimensionnelle est adaptée à l'interrogation et l'analyse des magasins de données, mais est inadéquate pour maintenir et gérer efficacement les données d'un entrepôt sur de longues périodes de temps. En effet, les nombreuses redondances sont engendrées par l'organisation multidimensionnelle des données.
- L'aspect temporel des données est peu étudié. Notamment, les données devenues inutiles dans l'entrepôt sont généralement supprimées. Or, aucune proposition ne fournit de mécanismes intermédiaires permettant par exemple d'archiver automatiquement les données détaillées à un niveau plus élevé exploitable par les décideurs.

5.5 Orientations de recherche

Nous proposons d'améliorer la prise de décision, aussi bien dans Assurance Maladie que dans les services médicaux des hôpitaux (par exemple en réanimation), en construisant un système décisionnel. L'architecture générale d'un tel système comporte des sources de données à partir desquelles l'entrepôt est alimenté par des extractions. L'exploitation des données pour les prises de décision est réalisée au travers d'outils de manipulation et d'interrogation.

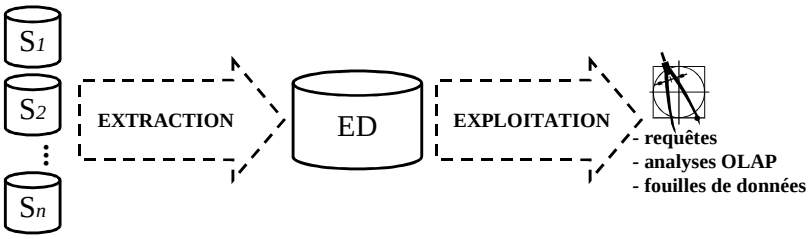


Figure 19 : Architecture générale des systèmes d'aide à la décision.

5.5.1 Distinction de l'entrepôt des magasins de données

A notre sens, **la conception d'un système décisionnel doit passer par la séparation de l'entrepôt de données et des magasins de données**. L'entrepôt proprement dit est le lieu centralisé de toute l'information pertinente pour les utilisateurs tandis que le magasin de données est un extrait d'entrepôt dédié à un type d'utilisateurs et répondant à un besoin spécifique. Ainsi l'entrepôt de données ne supporte pas directement les processus décisionnels (OLAP). Cette activité est réservée aux magasins qui améliorent les performances d'interrogation sans se soucier des redondances d'information ; chaque magasin stocke une partie de l'information disponible dans l'entrepôt afin de répondre à un objectif décisionnel précis pour un groupe d'utilisateurs ayant les mêmes besoins.

Ces **différences de fonction et d'objectif se répercutent dans la modélisation** de ces deux espaces de stockage.

- **L'entrepôt de données** vise à stocker l'information décisionnelle disponible dans l'entreprise et à maintenir cette information et ces évolutions au cours du temps. Il **n'est donc pas organisé de manière multidimensionnelle**.
- **Les magasins de données sont dédiés aux analyses décisionnelles de type OLAP**. **La modélisation multidimensionnelle est utilisée à ce niveau** puisqu'elle s'avère adaptée à ce type d'activité [Codd 1994] [Kimball 1996].

Le système décisionnel ne s'adresse pas à un seul type d'utilisateurs. Nous distinguons ceux qui conçoivent l'entrepôt de données et ceux qui les exploitent. Afin d'éviter les ambiguïtés, nous emploierons :

- le terme générique d'**administrateur(s)** pour désigner la (ou les) personne(s) chargée(s) de concevoir, de mettre en place et de maintenir le système décisionnel tandis que
- le terme d'**utilisateurs** désignera plutôt ceux qui exploitent le système décisionnel.

L'entrepôt de données et les magasins de données sont donc conçus par le (ou les) administrateur(s) informaticien(s). Ces derniers sont chargés de maintenir l'information décisionnelle de l'entreprise.

Les magasins de données s'adressent aux **utilisateurs non experts** en informatique et chargés d'effectuer des analyses et de prendre des décisions. Il s'agit généralement d'un petit nombre de personnes qui sont les décideurs de l'entreprise. Ces utilisateurs n'accèdent pas directement aux informations de l'entrepôt, mais utilisent les magasins de données pour effectuer des interrogations et des analyses OLAP au travers d'interfaces visuelles.

Cependant, il est aussi utile de pouvoir accéder directement à l'ensemble des données décisionnelles disponibles dans l'entreprise ; ceci est utile pour effectuer des analyses

ponctuelles ou bien des recherches en vue de la construction de nouveaux magasins. L'accès aux données de l'entrepôt s'adresse à des **utilisateurs experts** en informatique puisque l'interrogation de l'entrepôt s'effectue au travers de langages de manipulation de type SQL.

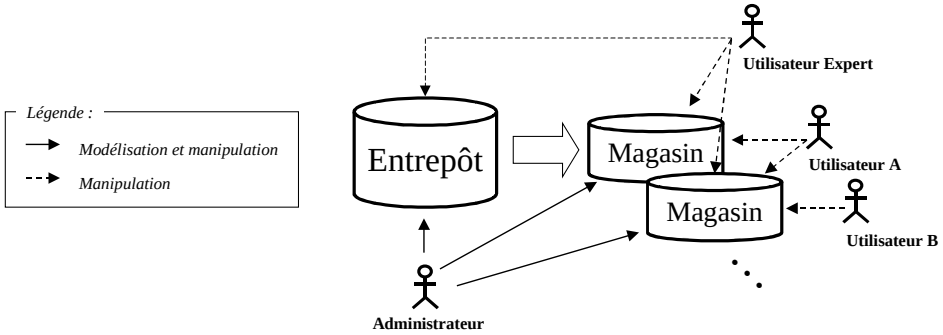


Figure 20 : Administration et utilisation des entrepôts et des magasins.

5.5.2 Notre architecture du système décisionnel

La mise en place d'un système décisionnel est une tâche complexe qui recouvre de nombreuses difficultés. Dans le but de définir des problématiques parfaitement identifiées et indépendantes les unes des autres, nous détaillons l'architecture générale à la Figure 21.

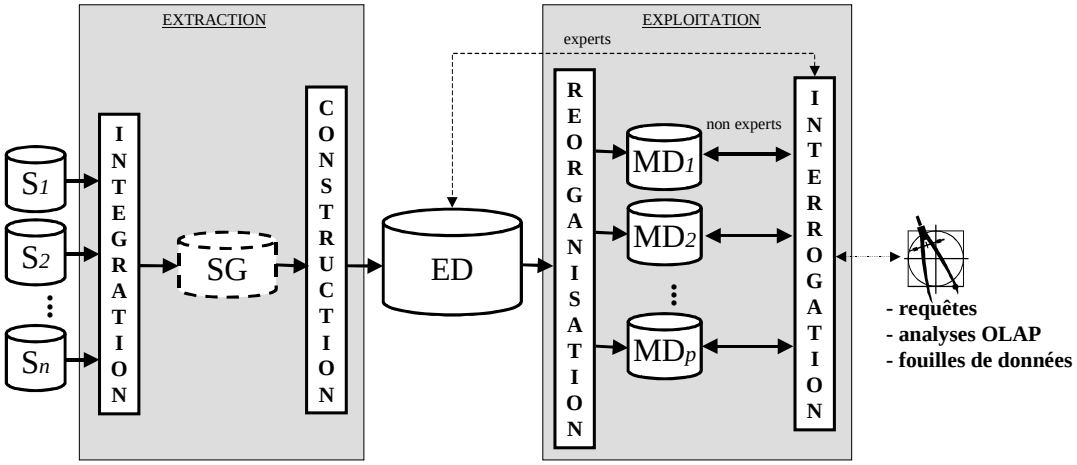


Figure 21 : Architecture détaillée des systèmes d'aide à la décision.

L'**intégration** se propose de résoudre les problèmes d'hétérogénéité (systèmes, modèles, formats et sémantiques des données,…) [Kedad, Métais 1999] des différentes sources de données en intégrant celles-ci dans une source globale. Cette source globale est virtuelle, c'est à dire que les données utilisées pour la décision restent stockées dans les sources de données et sont extraites uniquement au moment des mises à jour de l'entrepôt. La source globale est décrite au moyen du modèle de données orientées objet standard de l'ODMG [Cattel 1995]. Le choix du paradigme objet se justifie car il s'avère parfaitement adapté pour l'intégration de sources hétérogènes [Bukhres, Elmagarmid 1993] couramment utilisées dans le milieu médical [Pedersen, Jensen 1998]. L'intégration s'appuie sur des techniques de bases de données fédérées [Samos, et al 1998] et réparties comme celles qui ont été développées au sein de notre équipe [Ravat, Zurfluh 1995] [Ravat 1996].

La **construction** consiste à extraire les données pertinentes pour la prise de décision, puis à les recopier dans l'entrepôt de données, tout en conservant, le cas échéant, les changements

d'états des données. L'entrepôt de données constitue une collection centralisée, de données matérialisées et historiques (conservation des évolutions), disponibles pour les applications de l'entrepôt. Le modèle de l'entrepôt décrivant les données doit supporter des structures complexes [Pedersen, Jensen 1998, 1999] et supporter l'évolution des données au cours du temps [Inmon 1994] [Pedersen, Jensen 1999] [Yang, Widom 1998, 2000] [Mendelzon, Vaisman 2000].

La **réorganisation** permet de restructurer les données dans des magasins de données ; la réorganisation des données vise à supporter efficacement les processus d'interrogation et d'analyse tels que les applications OLAP ("*On-Line Analytical Processing*") [Codd 1993] et la fouille de données (*Data Mining*) [Fayyad, et al 1996] [Crimmins, et al 1998]. Pour ce faire, les données importées dans les magasins sont organisées de manière multidimensionnelle.

L'**interrogation** consiste à manipuler les données multidimensionnelles afin d'analyser les tendances passées pour prendre des décisions. Les données sont représentées sous une forme visant à faciliter leur compréhension et leur manipulation pour les décideurs non informaticiens (tableaux à *n* dimensions, graphiques,…).

Le Tableau 9 résume le rôle de chaque module constituant notre système décisionnel. A chacun de ces modules correspond des problématiques distinctes et spécifiques.

		Entrées	**Sorties**	**Problématiques**
INTEGRATION		$S_1,\ldots, S_n$	SG	Hétérogénéité (systèmes, modèles, formats et sémantiques des données), Distribution
CONSTRUCTION		SG	ED	Extraction des données, Modélisation de l'entrepôt, Maintenance de l'entrepôt, Historisation des données.
REORGANISATION		ED	$MD_1,\ldots,MD_p$	Extraction des données, Modélisation multidimensionnelle.
INTERROGATION	*expert*	ED	résultats	Manipulations temporelles, Manipulations multidimensionnelles, Interaction et présentation des résultats (graphes, statistiques,…)
	non expert	MD_i		

Tableau 9 : Les problématiques de recherche des systèmes d'aide à la décision.

5.5.3 Eléments de notre recherche

Dans cette thèse, nous étudions l'entrepôt de données, les magasins de données et leur interrogation.

1. Tout d'abord nous définissons **un modèle de données conceptuel objet dédié aux entrepôts** de données complexes, temporelles et archivées. Ce modèle intègre les concepts spécifiques d'objet entrepôt, de classe entrepôt et d'environnement.
2. De plus, nous formalisons **les processus d'extraction mis en jeu dans l'élaboration des structures de données de l'entrepôt** ainsi que des comportements de ces données. Nous modélisons l'extraction au travers de compositions de fonctions d'extraction.

Puis nous définissons des mécanismes d'interrogation permettant de manipuler les données utiles aux décideurs.

3. Nous proposons **une algèbre pour la manipulation des données de l'entrepôt** qui prend en compte les spécificités des objets entrepôt (évolutions détaillées et archivées) et permet d'exploiter l'évolution qu'ils décrivent. Nous définissons également un opérateur de transformation des objets entrepôt sous une forme chronologique de série temporelle ; cette transformation de l'évolution en période temporelle homogène facilite ainsi la compréhension des tendances et supporte les traitements automatiques d'analyse. Il s'agit d'une interrogation dédiée aux utilisateurs experts s'intéressant à un aspect particulier de l'information.
4. Nous proposons **une démarche de transformation des données dans des magasins multidimensionnels et une algèbre pour la manipulation de l'information contenue dans les magasins**. Les données multidimensionnelles sont extraites de l'entrepôt et stockées dans un magasin dédié aux analyses et à l'aide à la prise de décision. Il s'agit de l'interrogation dédiée aux utilisateurs décideurs (non experts) désirant avoir une vue globale de l'information.

Le modèle (1), le mécanisme d'extraction (2), l'interrogation de l'entrepôt (3) tandis d’un quatrième chapitre traite de la modélisation et de l’interrogation des magasins de données multidimensionnelles (4).

CHAPITRE II :
UN MODELE DE REPRESENTATION OBJET POUR UN ENTREPOT DE DONNEES EVOLUTIVES

1 INTRODUCTION A LA MODELISATION CONCEPTUELLE DES ENTREPOTS

1.1 Objectif

Nous avons défini une architecture, pour les systèmes d'aide à la décision, basée sur l'approche des entrepôts de données [Bret, Teste 1999] [Ravat, Teste 2000a]. Cette architecture comporte quatre composantes essentielles :

- l'intégration des sources de données dans une source globale virtuelle,
- la construction de l'entrepôt à partir d'extractions de données issues de la source globale,
- la réorganisation des données de l'entrepôt dans des magasins orientés décision,
- l'interrogation des magasins pour la prise de décision.

Les travaux présentés dans ce chapitre concernent la construction de l'entrepôt de données à partir d'une source globale. Plus précisément, l'objet de ce chapitre est de définir les concepts nécessaires à la description d'un entrepôt de données complexes et évolutives au travers d'un modèle conceptuel de données.

La source globale, à partir de laquelle l'entrepôt de données est construit, est obtenue par intégration des systèmes qui supportent l'activité de gestion des transactions courantes de l'entreprise (applications OLTP [Codd 1993]). Dans notre contexte, l'entrepôt de données constitue un référentiel centralisé servant de base aux processus de prise de décision ; l'entrepôt collecte l'ensemble des données (pertinentes pour aider les décideurs) disponibles au niveau de la source globale.

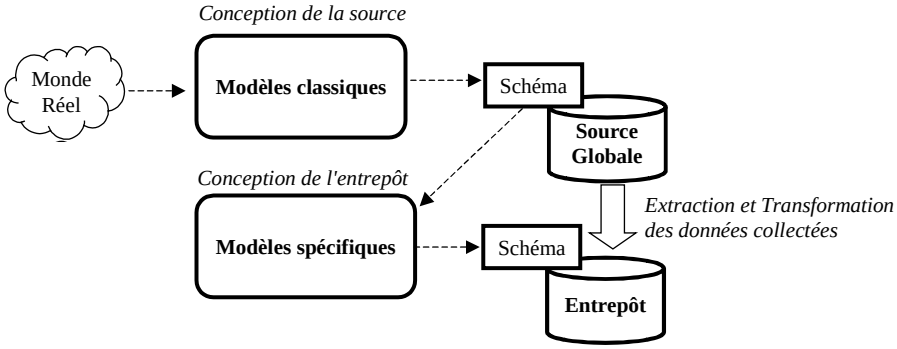


Figure 22 : Principe de la conception de l'entrepôt.

La source globale peut être représentée au travers d'un modèle classique dédié à la gestion des données transactionnelles (modèle relationnel, modèle E/A). Dans notre architecture du système décisionnel, nous décrivons la source globale avec le modèle standard de l'ODMG [Cattel 1995], étendu par la composition d'objets [Bertino, et al 1998b]. En effet, comme nous l'avons indiqué dans le chapitre précédent, des travaux récents prônent l'utilisation de l'objet pour résoudre les problèmes d'hétérogénéité entre les sources intégrées et pour faciliter l'interopérabilité en général [Burkhres, Elmagarmid 1993] [Ravat 1996].

Il est nécessaire de définir un modèle spécifique permettant de modéliser l'entrepôt à partir du schéma de la source globale. Ce modèle de données doit prendre en compte cinq caractéristiques.

1. **Gestion efficace des données décisionnelles**. Dans le but de garantir la pérennité de l'entrepôt, le modèle doit permettre la manipulation des données de manière efficace en limitant la redondance et en maintenant la cohérence des données.

2. **Gestion de données complexes**. Nos travaux se situent dans le contexte médical qui utilise couramment des structures de données complexes [Lapujade, Ravat 1997] [Pedersen, Jensen 1998, 1999]. De manière plus générale, les applications médicales utilisent des données à structures complexes qui nécessitent des modèles ayant une expressivité avancée et supportant une sémantique riche (par exemple, Lapujade et Ravat dans [Lapujade, Ravat 1997] ou bien Pedersen et Jensen dans [Pedersen, Jensen 1999] décrivent le dossier patient dans les hôpitaux comme une structure complexe composée d'un dossier d'admission, de dossiers médicaux, d'analyses,…).

3. **Gestion de l'origine des données et extraction**. Une caractéristique importante de l'entrepôt réside dans le fait qu'il est issu d'une source globale ; celle-ci intégrant l'ensemble des bases opérationnelles disponibles dans l'entreprise. L'entrepôt doit par conséquent être muni de mécanismes permettant :

   - l'extraction des seules données utiles pour les décideurs,
   - la répercussion dans l'entrepôt des mises à jours survenues dans les sources de données [Widom 1995] [Gupta, Mumick 1995].

   Cette dernière exigence est nécessaire pour que l'entrepôt corresponde à une vue actualisée de l'information disponible dans l'organisation (vue matérialisée [Gupta, Mumick 1995]).

4. **Gestion de l'historique des données**. Les décideurs utilisent les données actuelles mais également le passé des données (c'est à dire leurs évolutions au cours du temps) [Inmon 1994] [Chaudhuri, Dayal 1997] [Yang, Widom 1998, 2000] [Pedersen, Jensen 1999]. En outre, les sources de données qui sont souvent les bases de production de l'entreprise ne sont pas prévues pour conserver l'historique des données (chaque mise à jour écrase l'ancienne valeur) ou bien conservent les évolutions sur des périodes de temps insuffisantes pour les processus d'analyse décisionnelle (les bases dans les services hospitaliers ne conservent les données historiques que sur quelques mois tandis que les analyses OLAP nécessitent des historiques de plusieurs années [Inmon 1994]). Le modèle de l'entrepôt doit donc être muni de mécanismes évolués permettant de représenter l'information temporelle complexe.

5. **Mécanisme d'archivage**. Une difficulté importante, liée à la conservation des évolutions des données, est le volume des données engendré par le stockage des historiques [Inmon 1994]. De plus, au delà d'une certaine date, les décideurs n'ont généralement pas besoin du détail de toutes les évolutions ; c'est notamment le cas dans les applications médicales permettant par exemple de suivre les évolutions d'une épidémie. Il est inutile de conserver l'historique des données sous une forme exhaustive ; une conservation archivée (synthétique) est parfois suffisante voir même plus utile (car les décideurs s'appuient souvent sur une information reflétant les tendances et non pas le détail de l'activité).

1.2 Existant

Les travaux relatifs aux entrepôts de données[13] se concentrent pour l'essentiel sur les aspects multidimensionnels [Agrawal, et al 1995] [Li, Wang 1996] [Gyssen, Lakshmanan 1997] [Lehner, et al 1998] [Pedersen, Jensen 1999] et sur les problèmes concernant la gestion, la maintenance et la configuration des vues matérialisées [Gupta, Mumick 1995] [Huyn 1997] [Theodoratos, Sellis 1997] [Zhuge, et al 1998] [Kotidis, Roussopoulos 1999] [Yang, Widom 2000].

Tous ces travaux se situent dans un environnement relationnel tandis que les besoins des applications comme les applications médicales nécessitent une sémantique riche prenant en compte des données complexes [Pedersen, Jensen 1998]. Une solution consiste à définir un modèle orienté objet pour les entrepôts [Trujillo, Palomar 1998] [Pedersen, Jensen 1999] [Ravat, Teste 2000c]. Notamment, dans [Perdersen, Jensen 1999] les auteurs définissent un modèle orienté objet multidimensionnel pour gérer des dimensions complexes. En raison de l'organisation multidimensionnelle des données, cette solution engendre des redondances d'informations qui rendent difficiles une gestion efficace de l'entrepôt dans le temps. En outre, le modèle présenté dans [Perdersen, Jensen 1999] intègre des données temporelles mais ne propose pas de mécanismes d'archivage comme nous souhaitons le proposer.

Malgré l'aspect majeur des données temporelles dans les systèmes d'aide à la décision, peu de travaux traitent de la gestion du temps dans l'entrepôt. Plusieurs travaux sur les entrepôts se sont concentrés sur certains aspects liés aux données temporelles [Yang, Widom 1998, 2000] [Gupta, Mumick 1999]. Néanmoins ces travaux n'abordent pas les problèmes de conception et de modélisation des données temporelles dans l'entrepôt. Ils ne proposent pas un mécanisme d'archivage permettant de conserver les évolutions sous une forme adéquate pour les décideurs.

1.3 Proposition

Notre proposition consiste à définir un modèle conceptuel de données orienté objet intégrant des données temporelles et archivées [Teste 1999b] [Ravat, Teste 2000c] [Ravat, Teste, Zurfluh 2000b]. L'approche objet [Cattel 1995] lui confère une puissance d'expressivité et une riche sémantique. Une contribution de notre modèle est la définition d'un mécanisme d'archivage automatique intégrant des aspects temporels pour conserver l'évolution des données sous une forme adaptée aux besoins des décideurs.

Ce chapitre présente notre modèle et comporte trois sections.

- La section 2 définit le concept d'objet entrepôt qui intègre la valeur courante extraite de la source globale, mais également les valeurs anciennes. Ces valeurs historiques sont stockées de manière pertinente pour la prise de décision, soit sous une forme détaillée, soit sous une forme résumée (archivée).

[13] *Bibliographies disponibles sur Internet :*
- *http://www.cs.toronto.edu/~mendel/dwbib.html*,
- *http://www.informatik.tu-darmstadt.de/DVS1/staff/wu/dw.html*,
- *http://www-db.stanford.edu/warehousing/publications.html*.

- La section 3 décrit le concept de classe entrepôt qui étend le concept de classe standard défini par l'OMDG [Cattel 1995] pour prendre en compte les caractéristiques évolutives des objets entrepôt.
- La section 4 présente le concept d'environnement qui unifie les granularités d'historisation (attribut, classe, ensemble) et qui paramètre le comportement temporel des données.

2 OBJET ENTREPOT

Nous utilisons les conventions de notation suivantes : { } représente un ensemble, < > représente une liste.

2.1 Définition des objets et des états

Un **objet entrepôt** modélise une entité extraite de la source globale. Il peut être constitué à partir d'un ou plusieurs objets source ou d'une partie d'objets source. Dans la suite du développement, nous désignerons par le terme **origine** le ou les objets source à partir desquels un objet entrepôt est créé.

Un objet entrepôt est caractérisé par un **identifiant** interne, lequel est immuable, persistant au cours du cycle de vie de l'objet entrepôt et indépendant de celui de l'origine. L'entrepôt doit conserver l'évolution de valeur des objets, tandis que la source de données ne contient généralement que l'état courant, ou bien, ne conserve qu'une partie récente des évolutions. Par conséquent, dans un entrepôt, l'administrateur peut décider de conserver :

- l'image de son origine, c'est à dire son **état courant**,
- les états successifs que prend son origine dans le temps, c'est à dire ses **états passés**,
- un résumé de ses états passés successifs, c'est à dire l'agrégation de certains états passés, appelée **états archivés**.

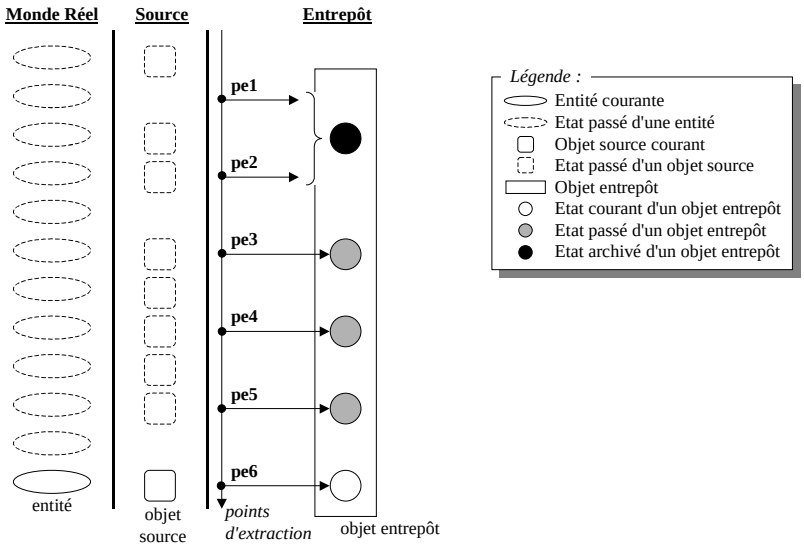


Figure 23 : Modélisation d'un objet entrepôt.

La Figure 23 illustre notre proposition de modélisation des objets entrepôt, en représentant un objet entrepôt possédant un état courant, trois états passés et un état archivé. Les états archivés correspondent à l'agrégation de certaines valeurs passées devenues inutiles sous une forme détaillée (les états passés devenus inutiles sont supprimés de l'entrepôt après l'archivage).

Nous définissons le concept d'objet entrepôt par l'expression suivante :

<u>*Définition*</u> :
Un **objet entrepôt** *o* est défini par un n-uplet (*oid*, S_0, *Histoire*, *Archive*, *Origine*) où

- *oid* est l'identifiant interne,
- S_0 est l'**état courant** correspondant à la dernière valeur extraite,
- *Histoire*=$\{S_{p1}, S_{p2},\ldots, S_{pp}\}$ est un ensemble fini d'**états passés** correspondant au détail des évolutions de valeur de l'objet entrepôt,
- *Archive*=$\{S_{a1}, S_{a2},\ldots, S_{aa}\}$ est un ensemble fini d'**états archivés** correspondant à une agrégation de certaines évolutions détaillées (états passés),
- *Origine*=$\{s\_oid_1, s\_oid_2,\ldots, s\_oid_s\}$ est un ensemble fini des identifiants des objets source à partir desquels la valeur de l'objet entrepôt est obtenue.

Les identifiants de l'origine (*Origine*) doivent être présents au niveau la source globale. Lorsque ceux-ci sont supprimés, l'objet entrepôt ne peut plus être rafraîchi : il est figé (*cf.* section 2.3).

Nous définissons le concept d'état par l'expression suivante :

<u>*Définition*</u> :
Un **état** S_i se définit par un couple ($DomT_i$, V_i) où

- $DomT_i$=<Int^1; …;Int^h> est le **domaine temporel** correspondant aux instants durant lesquels l'état S_i a été courant,
- V_i est la valeur des propriétés de l'objet durant les instants de $DomT_i$.

Le domaine temporel est modélisé par une liste d'intervalles. Nous définissons en détail les concepts de domaine temporel, d'intervalle, d'instant et de durée dans la section 2.2.

Un état est constitué d'une **valeur structurelle** (V_i) associée à une **valeur temporelle** ($DomT_i$). L'objet entrepôt regroupe plusieurs états.

- Un seul état courant enregistre la dernière valeur extraite pour chaque propriété de l'objet entrepôt.
- Des états passés correspondent aux anciennes valeurs des propriétés de l'objet. Ces états passés modélisent les évolutions détaillées des propriétés spécifiées comme temporelles (propriétés dont l'administrateur souhaite conserver les évolutions détaillées).
- Des états archivés agrègent les valeurs passées. Les états passés devenus inutiles pour les décideurs peuvent être agrégés, puis supprimés. Les états archivés modélisent les évolutions de l'objet sous une forme résumée. Seules les propriétés dont les évolutions détaillées sont conservées peuvent être archivées.

> <u>EXEMPLE</u> : Considérons l'exemple d'un objet entrepôt représentant le centre hospitalier Hôtel Dieu dans la ville de Paris ; la Figure 24 représente cet objet entrepôt. Pour simplifier l'expression des

dates, nous supposons que l'entrepôt est rafraîchi uniquement chaque année. Ces évolutions sont conservées de manière détaillées durant quatre années, puis elles sont archivées sur des durées de cinq années.

L'objet entrepôt est identifié de manière unique par l'identifiant *oid1*. Il comprend plusieurs états : un état courant, trois états passés et deux états archivés. Les différents états sont identifiés par le couple (*oid1*, *date*).

-L'état courant comprend cinq attributs ("*nom*", "*ville*", "*budget*", "*nb_services*", "*creation*").

-Les états passés ne comportent que les attributs ("*budget*", "*nb_services*") dont l'administrateur de l'entrepôt souhaite conserver les évolutions détaillées.

-Les états archivés sont constitués de l'attribut "*budget*" dont les évolutions détaillées obsolètes sont agrégées par période de cinq ans et d'un attribut système "*etats*" qui représente le nombre d'états passés archivés.

Chaque état est associé à son domaine temporel qui représente le temps au cours duquel la valeur de l'état a été ou est encore courante. Pour se rapprocher des références humaines, les instants qui bornent les intervalles de temps sont représentés par une date ; le symbole *now* désigne l'instant présent.

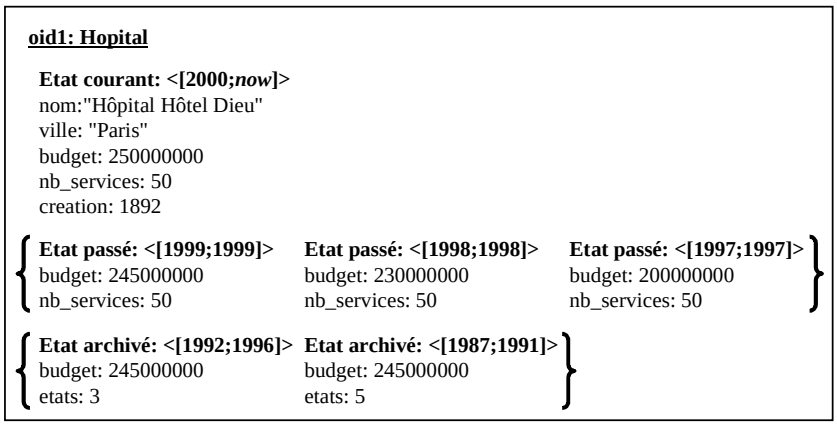


Figure 24 : Exemple d'un objet entrepôt.

2.2 Temps et domaine temporel

L'entrepôt de données contient des objets entrepôt formés de différents états. Chaque état représente une valeur structurelle associée à une valeur temporelle représentée au travers d'un domaine temporel. Cette section se propose de préciser la représentation du temps [Schwer, et al. 1995] dans l'entrepôt au travers d'un modèle temporel comportant des types temporels de base dont en particulier le domaine temporel qui permet de dater les différents états.

2.2.1 Modèle temporel linéaire discret

Dans le monde réel, l'espace temporel peut être vu comme une droite continue. Notre objectif est de proposer un modèle temporel, proche de la représentation humaine, qui appréhende le temps de manière empirique en utilisant des variables discrètes (par exemple le calendrier Grégorien).

Nous adoptons un modèle temporel **numérique**, **linéaire** et **discret** qui approche le temps de manière granulaire au travers d'**unités temporelles** d'observation [Wang, et al 1997].

- La numérisation apporte la puissance des opérations algébriques.
- La linéarité formalise la succession des événements dans le temps.
- La discrétisation fixe la précision de l'unité temporelle d'observation. Elle fournit la notion de durée au travers d'une fonction de distance.
- Les unités temporelles offrent différents niveaux de précision en fixant la sémantique des valeurs numériques.

La droite continue du temps est divisée en une suite de segments définissant une partition ; chaque point d'un segment est approché par le grain de temps qui le contient. Chaque unité temporelle correspond à une partition de la droite du temps.

<u>*Définition*</u> :
Une **unité temporelle** se définit par une fonction U de l'ensemble des entiers naturels ($\mathbb{N}$) vers l'ensemble des réels ($\mathbb{R}$), $U : \mathbb{N} \rightarrow \mathbb{R}$, présentant les propriétés suivantes :

- $\forall i \in \mathbb{N}, \forall j \in \mathbb{N}, i<j, U(i) \neq 0 \wedge U(j) \neq 0 \Rightarrow \forall r \in U(i), \forall r' \in U(j), r<r'$,
- $\forall i \in \mathbb{N}, \forall j \in \mathbb{N}, U(i)=0 \Rightarrow U(j)=0$.

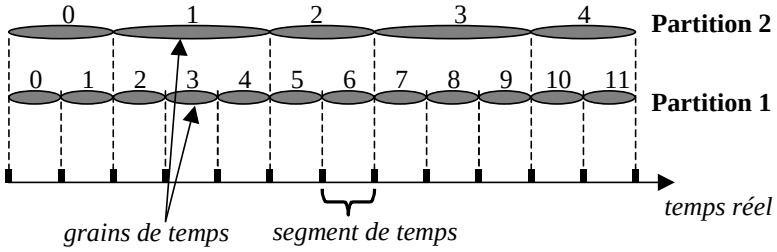


Figure 25 : Partition de la droite du temps.

Des exemples d'unités temporelles sont le *jour*, le *mois*, l'*année*, correspondant à une partition de la droite du temps. Ainsi, on définit l'unité temporelle *année* ayant pour origine l'année *1900* de la manière suivante : *année*(*0*) est le grain de temps correspondant à l'année *1900*, *année*(*1*) est le grain de temps correspondant à l'année *1901*, et ainsi de suite…

Nous définissons également une relation "*est plus fine que*" entre les unités temporelles.

<u>*Définition*</u> :
Soient deux unités temporelles U_1 et U_2. L'unité temporelle U_1 **est plus fine que** l'unité temporelle U_2, notée $U_1 \leq U_2$, si pour chaque entier i il existe un entier j tels que $U_1(i) \subseteq U_2(j)$.

Par exemple, l'unité *minute* est plus fine que l'unité *heure*, l'unité *heure* est plus fine que l'unité *jour*… Cependant la relation $\leq$ n'est pas un ordre total ; par exemple, les unités temporelles *semaine* et *mois* ne sont pas comparables (l'unité *semaine* n'est pas plus fine que *mois* et l'unité *mois* n'est pas plus fine que *semaine*) car une semaine peut être à cheval sur deux mois.

Un **calendrier** est un ensemble d'unités temporelles formant un treillis suivant la relation "*est plus fine que*".

2.2.2 Types temporels

Ce cadre formel modélisant le temps dans l'entrepôt permet de définir des types temporels. Ces types offrent une puissance suffisante pour décrire et manipuler les évolutions des données dans l'entrepôt.

2.2.2.1 Durée

Une durée est une quantité de temps relativement à une unité temporelle.

<u>*Définition*</u> :
Une **durée** *Dur* est définie par le couple (*q*, *U*) où

- *q* est un entier relatif désignant la quantité de temps,
- *U* est l'unité temporelle qui fixe la sémantique de la quantité temporelle.

Une durée n'est pas fixée sur l'axe temporel, c'est à dire qu'elle n'est pas positionnée par rapport à l'origine. Par exemple, *Dur*(*5*, *année*) est une quantité de temps constituée de cinq grains de temps consécutifs (sur l'unité temporelle *année*).

On pose *T_Unite* le type représentant les unités temporelles. Nous définissons deux fonctions associées au type temporel durée, noté *T_Duree*.

- *Unite* : *T_Duree*→*T_Unite* | *Unite*(*Dur*)=*U*. Cette fonction permet d'obtenir l'unité temporelle d'une durée.
- Distance : *T_Duree*→*T_Entier* | *Distance*(*Dur*)=*q*. Cette fonction permet d'obtenir la quantité de temps représentée par une durée. La sémantique de l'entier retourné est donnée par l'unité temporelle à laquelle on observe la durée.

2.2.2.2 Instant

Un instant *Ins*=(*n*, *U*) est un grain de temps relativement à une unité temporelle.

<u>*Définition*</u> :
Un **instant** *Ins* est défini par le couple (*n*, *U*) où

- *n* est un entier positif désignant le numéro du grain,
- *U* est l'unité temporelle qui fixe la sémantique du grain.

Par exemple, *Ins*(*0*,*année*) est l'instant correspondant au grain *0* de l'unité temporelle année ; il représente l'année *1900* pour un calendrier contenant l'unité temporelle *année* et dont l'origine est *1900*.

Nous définissons deux fonctions d'accès associées au type temporel instant noté *T_Instant*.

- *Unite* : *T_Instant*→*T_Unite* | *Unite*(*Ins*)=*U*
- *Point* : *T_Instant*→*T_Entier* | *Point*(*Ins*)=*n*

Notre modèle est muni d'un ensemble de fonctions de comparaison entre les instants {=, ≠, <, ≤, >, ≥}. Nous définissons à tout moment un instant particulier, noté *now*, représentant l'instant présent.

2.2.2.3 Intervalle

Un intervalle est un ensemble d'instants.

<u>*Définition*</u> :
Un **intervalle** *Int* est défini par le couple (*InsD*, *InsF*) où

- *InsD* est l'instant de début,
- *InsF* est l'instant de fin tels que *InsD*≤*InsF* et *Unite*(*InsD*)=*Unite*(*InsF*).

Un intervalle *Int* représente l'ensemble des instants contenus entre son instant de début et son instant de fin : *Int*={*Ins* | *InsD*≤*Ins*≤*InsF*}. Un intervalle est noté [*InsD* ; *InsF*].

Par exemple, [*Ins*(*3*, *année*);*Ins*(*6*, *année*]={*Ins*(*3*,*année*), *Ins*(*4*,*année*), *Ins*(*5*,*année*), *Ins*(*6*,*année*)}. Pour un calendrier contenant l'unité temporelle *année* et dont l'origine est l'année *1900*, cet intervalle désigne l'intervalle de temps entre *1903* et *1906*.

Nous définissons quatre fonctions associées au type temporel intervalle noté *T_Intervalle*.

- *Unite* : *T_Intervalle*→*T_Unite* | *Unite*(*Int*)=*Unite*(*InsD*)=*Unite*(*InsF*)
- *Debut* : *T_Intervalle*→*T_Instant* | *Debut*(*Int*)=*InsD*
- *Fin* : *T_Intervalle*→*T_Instant* | *Fin*(*Int*)=*InsF*
- *EstVide* : *T_Intervalle*→*T_Booleen* | *EstVide*(*Int*)=vrai ⇔ *Fin*(*Int*)<*Debut*(*Int*)

Notre modèle intègre également les treize opérateurs de Allen [Allen 1983] permettant de comparer les intervalles.

Schéma	Opérateurs	Opérateurs inverses
I J	I **b** J *(précède)*	J **bi** I *(suit)*
I J	I **m** J *(rejoint)*	J **mi** I *(est_rejoint)*
I J	I **o** J *(chevauche)*	J **oi** I *(est_chevauché)*
I J	I **s** J *(débute)*	J **si** I *(est_débuté_par)*
I J	I **d** J *(est_dans)*	J **di** I *(englobe)*
I J	I **f** J *(termine)*	J **f** I *(est_terminé_par)*
I J	I **e** J *(égale)*	-

Tableau 10 : Opérateurs de Allen.

2.2.2.4 Domaine temporel

Un domaine temporel est un ensemble ordonné d'intervalles de temps (le domaine temporel est équivalent au concept d'intervalle généralisé [Bestougeff, Ligozat 1989]).

<u>*Définition*</u> :
Un **domaine temporel** *DomT* est défini par $<Int^1; \ldots; Int^h>$ tel que $h \geq 1$. Les propriétés suivantes sont respectées :

- Chaque intervalle est non vide,
  $\forall k \in [1..h], \neg EstVide(Int^k)$,
- Les intervalles ont la même unité temporelle,
  $\forall k \in [1..h], \forall l \neq k \in [1..h], Unit(Int^k)=Unit(Int^l)$,
- Les intervalles sont disjoints,
  $\forall k \in [1..h], \forall l \neq k \in [1..h], Int^k \text{ b } Int^l \vee Int^l \text{ bi } Int^k$,
- Les intervalles sont ordonnés de manière non contiguë,
  $\forall k \in [1..h[, Fin(Int^k)<Debut(Int^{k+1})$.

Nous définissons quatre fonctions associées au type domaine temporel noté *T_DomTemporel*.

- *Unite* : $T\_DomTemporel \rightarrow T\_Unite \mid \forall k \in [1..h], Unite(DomT)=Unite(Int^k)=Unite(InsF)$,
- *Debut* : $T\_DomTemporel \rightarrow T\_Instant \mid Debut(DomT)=Debut(Int^1)$,
- *Fin* : $T\_DomTemporel \rightarrow T\_Instant \mid Fin(DomT)=Fin(Int^{hi})$,
- *Element* : $T\_DomTemporel \times T\_Entier \rightarrow T\_Intervalle \mid \forall k \in [1..h], Element(DomT, k)=Int^k$.

2.3 Statuts des objets entrepôt

Les objets entrepôt sont créés, sont supprimés et évoluent grâce aux opérations qui leur sont appliquées. L'évolution d'un objet est caractérisée par son cycle de vie. Le cycle de vie [*d*, *f*] d'un objet entrepôt recouvre les domaines temporels des états qui le composent :

$$< [d, f] >= \underbrace{DomT_0}_{\substack{\textit{domaine de} \\ \textit{l'état courant}}} \cup \underbrace{\left(\bigcup_{i=p1}^{pp} DomT_i\right)}_{\substack{\textit{union des domaines} \\ \textit{des états passés}}} \cup \underbrace{\left(\bigcup_{j=a1}^{aa} DomT_j\right)}_{\substack{\textit{union des domaines} \\ \textit{des états archivés}}}$$

Le cycle de vie sous entend un graphe des états possibles par lequel l'objet entrepôt doit passer ; nous parlerons de **statut** afin d'éviter toute confusion. Nous définissons deux statuts possibles pour les objets entrepôt :

- Un objet entrepôt est **actif** lorsque tous les objets source dont il dérive sont présents dans la source globale. Tant que l'objet entrepôt est actif, il est possible de le faire évoluer en répercutant les mises à jour survenues dans la source globale.
- Un objet entrepôt est **figé** lorsqu'au moins un objet source dont il est issu a été supprimé de la source globale. Lorsque l'objet entrepôt est figé, cela signifie qu'il n'est plus possible de le rafraîchir puisqu'il manque au moins une partie de l'information source nécessaire.

Lorsque l'objet est actif, $Fin(DomT_0)=now$ et lorsque l'objet est figé, $Fin(DomT_0)=f$.

2.4 Rafraîchissement des objets entrepôt

Deux approches sont envisageables pour rafraîchir l'entrepôt de données [Widom 1995], c'est à dire pour répercuter dans l'entrepôt les évolutions qui surviennent au niveau de la source :

- l'approche du rafraîchissement périodique consiste à mettre à jour régulièrement l'entrepôt à des points d'extraction correspondant à un état de cohérence de la source (terminaison de toutes les transactions) ;
- l'approche du rafraîchissement incrémental consiste à répercuter chaque modification d'une valeur de la source dès que possible. Cette approche fait l'objet de nombreux travaux [Gupta, Mumick 1995] [Huyn 1997] [Zhuge, et al 1998] concernant la maintenance incrémentale et l'auto-maintenance des vues matérialisées.

Nous adoptons l'approche du **rafraîchissement périodique** puisque, selon notre contexte d'application (contrôle et suivi de l'activité des professionnels de santé et des patients), il est admis que l'analyse de l'activité des hôpitaux peut s'effectuer avec un décalage par rapport à la réalité.

Le rafraîchissement périodique est insensible à l'évolution réelle des entités : entre deux rafraîchissement l'entrepôt n'est pas mis à jour ; or il peut y avoir plusieurs évolutions au niveau de la source. Ceci oblige l'administrateur à choisir une période de rafraîchissement de l'entrepôt adaptée aux besoins des applications décisionnelles et au rythme d'évolution de la source, tout en maintenant les performances de l'entrepôt lors de l'interrogation.

2.5 Synthèse

Nous avons défini le concept d'objet entrepôt qui intègre la valeur courante de l'objet (état courant) ainsi que les évolutions successives de l'objet, soit sous forme détaillée (états passés), soit sous forme résumée (états archivés). L'apport principal de notre proposition réside dans la capacité du modèle à définir des évolutions de valeurs répondant aux exigences des décideurs. Ces évolutions sont considérées au travers d'un modèle temporel linéaire discret.

3 CLASSE ENTREPOT

Les objets entrepôt ayant une même structure et un même comportement sont regroupés dans une classe.

3.1 Définition des classes entrepôt

Le contexte d'entrepôt de données ainsi que le concept d'objet entrepôt que nous avons définis induisent des extensions du concept standard de classe tel qu'il est défini dans [Cattel 1995].

- Les valeurs conservées par les objets entrepôt sont collectées à partir de la source globale ; par conséquent, il convient de définir le mécanisme d'extraction mis en jeu afin de préciser la correspondance entre les objets entrepôt et leur origine.

- Le concept d'objet entrepôt englobe le caractère évolutif des données en conservant leurs évolutions. Il est donc nécessaire de spécifier les propriétés temporelles et archivées des classes de l'entrepôt.

Nous définissons le concept de classe entrepôt par l'expression suivante :

<u>*Définition*</u> :
Une **classe entrepôt** *c* est définie par un n-uplet (Nom^c, $Type^c$, $Super^c$, $Extension^c$, $Mapping^c$, $Tempo^c$, $Archi^c$) où

- Nom^c est le nom de la classe,
- $Type^c$ est le type de la classe,
- $Super^c$ est l'ensemble des super-classes de c,
- $Extension^c=\{o_1, o_2,\ldots, o_x\}$ est l'ensemble fini d'objets entrepôt regroupés dans la classe *c*,
- $Mapping^c$ est la **fonction de construction** qui caractérise le processus d'extraction à partir duquel la classe *c* est générée et alimentée en instances,
- $Tempo^c$ est le **filtre temporel** qui caractérise l'ensemble des propriétés temporelles de la classe dont les évolutions de valeur sont conservées de manière détaillée (les propriétés temporelles définissent la structure des états passés de la classe),
- $Archi^c$ est le **filtre d'archives** qui caractérise l'ensemble des attributs archivés de la classe dont les évolutions sont conservées de manière agrégée (les attributs archivés définissent la structure des états archivés de la classe).

De manière classique, le nom de la classe permet d'identifier cette classe. En outre, les classes entrepôt sont organisées suivant une hiérarchie d'héritage, modélisée par $Super^c$. Nous retenons l'héritage par spécialisation des types et par inclusion d'objets [Atkinson, et al 1989] : la classe entrepôt c_i est sous classe entrepôt de c_j (c_j est super classe de c_i) notée $c_i \preccurlyeq c_j$ si et seulement si $Type^{ci} \supseteq Type^{cj}$ et $Extension^{ci} \subseteq Extension^{cj}$.

La fonction $Mapping^c$ permet de préciser comment est construite la classe entrepôt et fait l'objet d'une étude détaillée dans le chapitre III.

Dans la section 3.2 nous proposons une taxonomie des propriétés et opérations des classes entrepôt. Cette taxonomie est fondée sur l'origine de chaque propriété et opération. La section 3.3 détaille les concepts de filtres temporel et d'archives associés aux classes entrepôt.

3.2 Taxonomie des propriétés et des opérations

Le type $Type^c$ associé à une classe entrepôt *c* décrit le schéma, c'est à dire la structure et le comportement commun aux objets. Il est défini par le couple ($Structure^c$, $Comportement^c$) où

- $Structure^c=(Attribut^c, Relation^c)$ est un ensemble de propriétés correspondant aux attributs et relations sémantiques (association, composition) entre la classe *c* et une (ou plusieurs) autre(s) classe(s),
- $Comportement^c$ est un ensemble d'opérations applicables aux objets de la classe *c*.

<u>Remarque</u> :
Nous adoptons les termes d'opération, de propriété, d'attribut et de relation comme ils sont introduits par la standardisation de l'ODMG [Cattel 1995]. Le comportement d'un objet est

défini par l'ensemble des **opérations** que l'on peut appliquer sur un objet de son type. L'état d'un objet est défini par les valeurs assignées à un ensemble de **propriétés**. Celles-ci peuvent être ou bien des **attributs** de cet objet, ou bien des **relations** entre ce dernier et un ou plusieurs autres objets.

Dans le contexte des entrepôts de données, l'origine des propriétés et des opérations est une caractéristique très importante pour la gestion des données, notamment lors de leur rafraîchissement. Notre modèle intègre une taxonomie des propriétés et des opérations relative à leur origine.

Nous définissons plusieurs catégories d'attributs, de relations et d'opérations.

- Un **attribut dérivé** représente dans l'entrepôt un attribut directement issu de la source. Le type et la valeur d'un attribut dérivé sont identiques à ceux de l'attribut source.
- Un **attribut calculé** est un attribut dont la valeur est le résultat d'une fonction de calcul appliquée sur la source. Son type correspond à celui du résultat de la fonction. Ce mécanisme permet ainsi à l'administrateur d'agréger des données source dans l'entrepôt ; autrement dit, les attributs calculés représentent explicitement dans l'entrepôt une information stockée de manière implicite dans la source.
- Toute l'information utile aux décideurs n'est pas nécessairement contenue dans les sources de données mais peut être directement introduite par les utilisateurs via des attributs dits spécifiques. Un **attribut spécifique** n'est pas issu de la source globale, mais il est saisi par l'administrateur qui spécifie son type. Un tel attribut est valorisé soit par l'administrateur, soit par les utilisateurs de l'entrepôt qui peuvent ainsi compléter l'information décisionnelle contenue dans l'entrepôt.

A l'instar des attributs, nous définissons des **relations dérivées** et des **relations spécifiques**. Une relation dérivée représente une relation source ; elle est redéfinie entre les classes entrepôt issues des classes source impliquées dans la relation source dérivée. Une relation spécifique est une relation introduite par l'administrateur.

De même, nous définissons des **opérations dérivées** et des **opérations spécifiques**. Une opération dérivée est issue de la source globale. Cette opération dérivée d'une opération source est redéfinie au niveau de l'entrepôt en fonction des éléments présents. Une opération spécifique est une opération ajoutée par l'administrateur en fonction des besoins des utilisateurs. L'administrateur spécifie entièrement l'opération (signature et corps). Ainsi, il peut compléter le comportement de l'entrepôt pour satisfaire aux exigences liées aux applications décisionnelles.

Pour résumer, les éléments dérivés et calculés contenus dans l'entrepôt sont valorisés à partir de la source globale ou issus de celle-ci tandis que les éléments spécifiques ne sont pas issus de la source mais ils sont ajoutés par l'administrateur, permettant ainsi aux utilisateurs de compléter l'information contenue dans l'entrepôt.

Ainsi les attributs, les relations et les opérations d'un type noté $Type^c$ peuvent être définis par :

$Attribut^c = AttributDerive^c \cup AttributCalcule^c \cup AttributSpecifique^c$

$Relation^c = RelationDerive^c \cup RelationSpecifique^c$

$Comportement^c = ComportementDerive^c \cup ComportementSpecifique^c$

EXEMPLE : Considérons la classe entrepôt "*Hopital*" contenant l'objet entrepôt *oid1* présenté dans l'exemple précédent de la section 2.1. Nous décrivons le type de cette classe en utilisant le langage de définition standard de l'ODMG [Cattel 1995], étendu par la composition d'objet [Bertino, et al 1998b], et intégrant la taxonomie des propriétés et des opérations présentée ci-dessus : le préfixe 'D_' désigne un élément dérivé (attribut, relation ou opération), le préfixe 'S_' désigne un élément spécifique et le préfixe 'C_' désigne un attribut calculé.

```
Interface Hopital {
   D_attribute String nom;
   D_attribute String ville;
   D_attribute Float budget;
   C_attribute Short nb_services;
   S_attribute Short creation;
   }
```

La classe est constituée de trois attributs dérivés ("*nom*", "*ville*", "*budget*"), d'un attribut calculé ("*nb_services*") et d'un attribut spécifique ("*creation*").

Notre proposition semble induire une violation du principe de non volatilité défini pour les entrepôts de données par Inmon dans [Inmon 1994]. En effet, la non volatilité interdit aux utilisateurs de modifier les valeurs stockées dans un entrepôt. Or dans notre approche, seule la partie spécifique de l'entrepôt est modifiable ; le reste de l'entrepôt (parties dérivées et calculées) n'est pas modifiable par les utilisateurs respectant ainsi le principe de non volatilité. Cette approche confère à l'entrepôt une grande adaptabilité ; par exemple, dans notre contexte d'application, les médecins peuvent intégrer des informations confidentielles dans l'entrepôt à partir des consultations qu'ils effectuent.

3.3 Filtres temporels et d'archives

Le concept d'objet entrepôt englobe le caractère évolutif des données en conservant leurs évolutions sous une forme adaptée aux exigences des décideurs ; les objets entrepôt sont composés d'un état courant mais aussi d'états passés et d'états archivés.

La spécificité des objets entrepôt nécessite l'extension de la définition des classes telle qu'elle est proposée par l'ODMG. Dans notre contexte, le type de la classe définit la structure et le comportement de l'état courant. Nous complétons cette définition par les concepts suivants :

- un filtre temporel $Tempo^c$ définit la structure des états passés,
- un filtre d'archives $Archi^c$ définit la structure des états archivés.

Le **filtre temporel** $Tempo^c=\{(p_1, f_1), (p_2, f_2),\ldots, (p_t, f_t)\}$ caractérise les propriétés temporelles d'une classe entrepôt. Il est constitué d'un ensemble de couples (p_j, f_j) où

- p_j est une propriété temporelle et
- f_j est soit un attribut, soit une relation, soit une opération retournant un résultat (fonction) appartenant au type de la classe.

Les évolutions de valeur détaillées des propriétés temporelles du filtre sont conservées au travers d'états passés. S'il s'agit d'une opération, les évolutions de son résultat sont conservées à chaque point d'extraction (rafraîchissement de la classe entrepôt).

EXEMPLE : Nous complétons l'exemple précédent de la classe entrepôt "*Hopital*". Cette classe contient des objets entrepôt dont l'administrateur souhaite conserver certaines évolutions au travers d'états passés. Il s'agit de stocker les évolutions détaillées des attributs "*budget*" et "*nb_services*". La description du type de la classe entrepôt est alors complétée par le filtre temporel suivant :

```
Interface Hopital {
   D_attribute String nom;
   D_attribute String ville;
   D_attribute Float budget;
   C_attribute Short nb_services;
   S_attribute Short creation;
   }
with temporal filter {(budget, budget), (nb_services, nb_services)}
```

Le **filtre d'archives** $Archi^c=\{(a_1, f_1), (a_2, f_2),\ldots, (a_s, f_s)\}$ caractérise les propriétés archivées de la classe entrepôt. Il est constitué d'un ensemble de couples (a_j, f_j) où a_j est un attribut et f_j est une fonction d'agrégation. L'ensemble des attributs archivés est un sous ensemble des attributs temporels. Chaque attribut archivé est associé à une fonction d'agrégation qui définit la manière dont sont résumées les valeurs temporelles.

EXEMPLE : Nous poursuivons l'exemple de la classe entrepôt "*Hopital*". L'administrateur désire archiver les états passés devenus obsolètes. Seul l'attribut "*budget*" est archivé tandis que l'attribut "*nb_services*" ne l'est pas ; les états passés archivés sont supprimés de l'entrepôt. L'administrateur utilise la fonction moyenne pour agréger les valeurs passées du budget des hôpitaux. La description du type de la classe entrepôt est alors complétée par le filtre d'archives suivant :

```
Interface Hopital {
   D_attribute String nom;
   D_attribute String ville;
   D_attribute Float budget;
   C_attribute Short nb_services;
   S_attribute Short creation;
   }
with temporal filter {(budget, budget), (nb_services, nb_services)},
     archive filter {(budget, avg(budget))}
```

Deux archivages sont possibles.

- L'**archivage fort** est basé sur les fonctions d'agrégation de sommation, dénombrement, moyenne, maximum, minimum (*sum*, *count*, *avg*, *max*, *min*). Il consiste à résumer l'ensemble les états passés dans un seul état archivé. L'intérêt de cette approche est de réduire fortement le volume des données mais en provoquant la perte de la notion d'évolution.
- L'**archivage modéré** est basé sur les fonctions d'agrégation associées à une unité temporelle. Il consiste à résumer les états passés dans plusieurs états archivés, chacun correspondant à une période de temps (définie par une unité temporelle et une quantité de temps). L'intérêt de cette approche est de conserver l'évolution des données à un niveau de

détail plus élevé, ce qui limite le volume des données mais de manière moins importante que par l'approche précédente.

EXEMPLE : Pour illustrer la différence entre ces deux types d'archivage, la Figure 26 décrit un objet entrepôt de la classe "*Hopital*".

La Figure 26(a) correspond au filtre d'archives : `archive filter {(budget, avg(budget))}`.

La Figure 26(b) correspond au filtre d'archives : `archive filter {(budget, avg(budget))} with T_Unit(annee) by 5`

Cette dernière définition indique que les états passés archivés sont regroupés par des périodes de cinq ans ; chaque état archivé synthétise les évolutions détaillées survenues pendant cinq années.

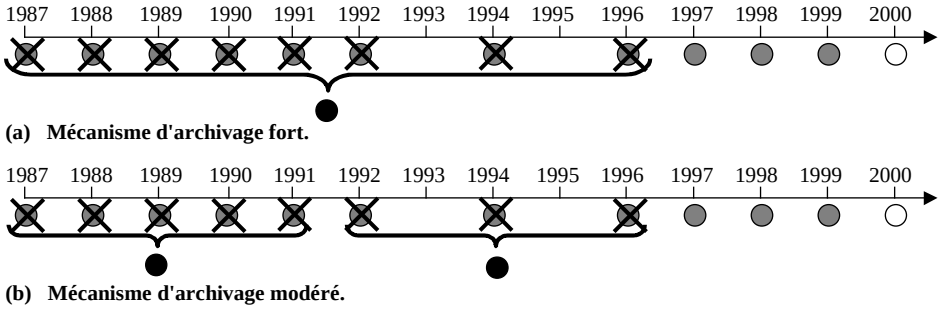


Figure 26 : Principe du mécanisme d'archivage.

En outre, notre modèle permet d'archiver un même attribut temporel selon différentes fonctions d'agrégation. Par exemple, il est possible d'archiver les évolutions du budget d'un hôpital en utilisant la fonction moyenne et la fonction maximum.

EXEMPLE : Nous reprenons l'exemple de la classe entrepôt "*Hopital*". Si l'administrateur souhaite archiver l'attribut "*budget*" en utilisant les fonctions d'agrégation moyenne et maximum. La description du type de la classe entrepôt est alors complétée par le filtre d'archives suivant :

```
Interface Hopital {
   D_attribute String nom;
   D_attribute String ville;
   D_attribute Float budget;
   C_attribute Short nb_services;
   S_attribute Short creation;
   }
with temporal filter {(budget, budget), (nb_services, nb_services)},
     archive filter {(budgetAVG, avg(budget)), (budgetMAX, max(budget))}
```

3.4 Synthèse

Nous avons défini le concept de classe entrepôt qui étend le concept de classe tel qu'il est défini par l'ODMG. Cette extension comprend :

- un filtre temporel qui définit les propriétés temporelles dont le détail des évolutions de valeur sera conservé par des états passés,
- un filtre d'archives qui définit des propriétés dont les évolutions de valeur seront résumées par des états archivés.

En outre, nous gérons de manière spécifique les données de l'entrepôt en fonction de leur origine. Pour cela, nous avons défini une taxonomie des attributs, des relations et des opérations.

4 ENVIRONNEMENT ET ENTREPOT

Il n'est pas nécessaire de conserver les évolutions de l'ensemble des données de l'entrepôt. L'administrateur doit pouvoir définir des parties temporelles dont la taille est en adéquation avec les exigences des applications décisionnelles afin de ne conserver que les évolutions pertinentes.

4.1 Définition des environnements

La modélisation flexible des données (évolutions détaillées ou résumées) contenues dans l'entrepôt induit de nouveaux problèmes.

- **Granularités d'historisation**. Dans l'entrepôt, l'administrateur doit pouvoir historiser un attribut pour les différents objets d'une classe (granularité attribut), les objets eux-mêmes (granularité classe) ou bien plusieurs classes et leurs relations (granularité ensemble).
- **Hétérogénéité du rafraîchissement**. Nous avons décidé de rafraîchir l'entrepôt de manière périodique. Néanmoins, ce rafraîchissement n'est pas uniforme dans un entrepôt, les données peuvent être mises à jour selon des périodicités différentes (jour, semaine, mois…) et variables dans le temps (congés annuels d'été, épidémies de grippes en hiver,…).
- **Hétérogénéité des filtres des classes entrepôt**. Les classes entrepôt archivent les évolutions détaillées suivant des critères d'archivage différents. Autrement dit, dans un même entrepôt, les évolutions détaillées sont conservées durant des périodes de temps variables suivant la classe d'objets entrepôt considérée.
- **Intégrité des relations sémantiques temporelles**. Lorsqu'une relation sémantique (association ou composition) reliant des classes entrepôt est historisée (conservation de ses évolutions détaillées via des états passés), il est nécessaire de conserver les évolutions détaillées des objets entrepôt reliés.

Nous proposons le concept d'**environnement** [Ravat, Teste, Zurfluh 1999], permettant de définir des parties temporelles homogènes, cohérentes et configurables.

<u>*Définition*</u> :
Un **environnement** *env* est défini par (Nom^{env}, C^{env}, $Config^{env}$) où

- Nom^{env} est le nom de l'environnement,
- $C^{env}=\{c_{e1}, c_{e2},\ldots,c_{en}\}$ est l'ensemble fini des classes contenues dans l'environnement,
- $Config^{env}$ est l'ensemble des paramètres de configuration, visant à définir différents paramètres locaux à l'environnement, caractérisant son comportement temporel et garantissant sa cohérence (période de rafraîchissement, critère d'archivage des états passés des classes,…).

L'environnement généralise l'historisation des données dans un entrepôt (il constitue l'unique moyen pour définir les données historisées). Ainsi, les objets entrepôt des classes appartenant à un environnement ont leurs évolutions conservées, tandis que les objets entrepôt des classes non contenues dans un environnement sont constituées uniquement de l'état courant ; $\forall c \in C^{ED}, \neg(\exists env \mid c \in C^{env}) \Rightarrow \forall o \in Extension^{c}, o.Histoire=\emptyset \wedge o.Archive=\emptyset$ (C^{ED} représente l'ensemble des classes définies sur un entrepôt *ED*).

> *EXEMPLE :* Nous reprenons l'exemple précédent de la classe entrepôt "*Hopital*". Les objets entrepôt regroupés dans cette classe sont historisés, c'est à dire que l'administrateur désire conserver les évolutions de certains attributs de la classe. La classe doit par conséquent appartenir à un environnement que nous décrivons par l'expression suivante :
>
> ```
> Environment Evolution { Hopital };
> ```
>
> Il est alors possible à l'administrateur de spécifier finement l'historisation de cette classe en définissant un filtre temporel et éventuellement un filtre d'archives.

L'environnement est identifié par son nom. De plus, chaque environnement décrit une partie temporelle de l'entrepôt caractérisée par des paramètres de configuration. Les environnements ne doivent pas englober une même classe entrepôt et ce, afin d'éviter des incohérences dans la configuration des données qu'ils contiennent. Ainsi, les environnements sont disjoints ; $\forall env_1 \in Env^{ED}, env_2 \in Env^{ED}, C^{env1} \cap C^{env2}=\emptyset$ (Env^{ED} représente l'ensemble des environnements définis sur un entrepôt *ED*).

4.2 Configuration des environnements

La configuration de chaque environnement ($Config^{env}$) a pour objectif de spécifier le comportement temporel de celui-ci. Les configurations sont des contraintes spécifiées par l'administrateur qui paramètre le comportement temporel des environnements. Elles déterminent :

- la gestion des objets entrepôt (périodicité des rafraîchissements, conservation des objets entrepôt figés…),
- la gestion des états temporels et archivés (critère d'archivage…).

Nous définissons les configurations d'environnement au travers d'un langage de configuration, basé sur l'approche ECA (Evènement-Condition-Action) [Dayal, et al. 1988] [Widom 1996]. Plus précisément, la définition par l'administrateur d'une règle de configuration respectera la syntaxe suivante :

```
<définition_regle>::= [create] configuration <nom_de_la_règle>
                       on <portée_de_la_règle>
                       when <événements>
                      [if <conditions>]
                       then <actions>
```

La clause <*nom_de_la_règle*> précise le nom de la règle de configuration. Dans un même environnement, chaque nom d'une règle sert d'identifiant.

La clause <*portée_de_la_règle*> définit la portée de la configuration. Il s'agit de préciser le nom de l'environnement auquel s'applique la règle.

La clause <*évènements*> dénote la (les) cause(s) qui déclenche(nt) l'exécution des instructions liées à la règle de configuration. Les événements possibles sont soit des événements temporels, soit des événements liés à l'invocation d'une opération.

La clause <*conditions*> est une expression booléenne ou une requête. Lorsque l'expression booléenne est vraie, les instructions liées à la configuration sont déclenchées. Lorsque la condition est une requête, la configuration s'applique sur l'ensemble des objets sélectionnés.

La clause <*actions*> représente l'ensemble des instructions qui doivent être exécutées. Les instructions précisent les actions qui sont appliquées lorsque la configuration est déclenchée.

> EXEMPLE : Nous complétons la définition de l'environnement "*Evolution*" de l'exemple précédent. Il s'agit de définir les règles de configuration qui caractérisent le comportement temporel de l'environnement et qui garantissent sa cohérence.
>
> L'administrateur définit ainsi une première règle de configuration relative à la période de rafraîchissement de l'environnement. Les évolutions survenues au niveau de la source globale sont répercutées chaque année.
>
> ```
> create configuration RAFRAICHIR on Evolution
> when T_Duree(now-self.Last_Refresh(), annee) > T_Duree(1, annee)
> then self.Refresh();
> ```
>
> Le symbole "*self*" représente l'élément de la portée sur lequel est appliqué la configuration ; dans cette configuration, "*self*" désigne l'environnement "*Evolution*". L'opération "*Last_Refresh()*" retourne l'instant du dernier rafraîchissement appliqué à l'environnement.
>
> L'administrateur ajoute une seconde configuration relative au critère d'archivage. Cette règle consiste à déterminer les états temporels à archiver ; autrement dit, la configuration caractérise le critère de pertinence ou de non pertinence des états passés de la classe "*Hopital*" de l'environnement "*Evolution*". Les états passés sont conservés durant quatre ans, puis sont archivés conformément au filtre d'archivage.
>
> ```
> create configuration ARCHIVAGE on Evolution
> when self.Refresh()
> if select E
>  from C in self.Classes(),
>  O in C.Extension(),
>  E in O.Histoire()
>  where T_Duree(now-Debut(E.DomT()), annee) > T_Duree(4, annee)
> then E.Archive(O);
> ```

4.3 Granularités d'historisation

Le concept d'environnement permet d'unifier l'historisation en offrant trois niveaux, appelés **granularités d'historisation** : granularité attribut, granularité classe, granularité ensemble.

4.3.1 Granularité classe

La granularité classe consiste à conserver les évolutions de chaque objet pour l'ensemble des valeurs des attributs de la classe (ainsi que des relations d'association et de composition réflexives). Pour ce faire, l'administrateur doit définir un environnement englobant

uniquement la classe à historiser. Son filtre temporel contient l'ensemble des attributs de la classe, garantissant ainsi la conservation de leurs évolutions détaillées au travers d'états passés.

> *EXEMPLE :* Considérons un environnement "*Niv_Classe*" constitué de la classe "*Hopital*". Il se définit de la manière suivante :
>
> ```
> Environment Niv_Classe { Hopital };
> ```
>
> Il s'agit de conserver les évolutions détaillées de l'ensemble des attributs de la classe "*Hopital*". Le filtre temporel de la classe regroupe l'ensemble des attributs.
>
> ```
> Interface Hopital {
>  D_attribute String nom;
>  D_attribute String ville;
>  D_attribute Float budget;
>  C_attribute Short nb_services;
>  S_attribute Short creation;
>  }
> with temporal filter {(nom, nom), (ville, ville), (budget, budget),
>  (nb_services, nb_services), (creation, creation)}
> ```

4.3.2 Granularité attribut

La granularité la plus fine est la granularité attribut qui consiste à conserver les évolutions d'un ou plusieurs attributs d'une classe. Pour cela, l'administrateur définit un environnement englobant uniquement la classe ; il définit le filtre temporel de la classe contenant seulement les attributs à historiser.

> *EXEMPLE :* Dans l'exemple précédent, on constate qu'il n'est pas judicieux d'historiser certains attributs tels que les attributs "*nom*" ou "*ville*". L'administrateur peut donc choisir la granularité attribut en construisant un environnement "*Niv_Attribut*" sur la classe "*Hopital*" et en spécifiant le filtre temporel suivant :
>
> ```
> Environment Niv_Attribut { Hopital };
>
> Interface Hopital {
>  D_attribute String nom;
>  D_attribute String ville;
>  D_attribute Float budget;
>  C_attribute Short nb_services;
>  S_attribute Short creation;
>  }
> with temporal filter {(budget, budget), (nb_services, nb_services)}
> ```

4.3.3 Granularité ensemble

La granularité ensemble (de classes) consiste à conserver les évolutions de plusieurs classes d'objets entrepôt ainsi que des relations d'association et de composition qui les relient. Le filtre temporel de chacune des classes historisées contient l'ensemble des attributs et des relations dont les évolutions détaillées sont conservées.

Les relations sémantiques historisées (appartenant aux filtres temporels) sont spécifiées parmi les relations liant les classes de l'environnement lui-même ; les relations reliant des classes externes à l'environnement ne peuvent en aucun cas être historisées.

> *EXEMPLE :* Nous considérons ici la classe entrepôt "*Hopital*" ainsi que sa classe composante "*Service*" représentant les services hospitaliers qui constituent chaque hôpital. Ces deux classes sont reliées par une relation sémantique de composition. Un administrateur souhaitant conserver les évolutions de cette relation pour suivre l'évolution détaillée de l'organisation des hôpitaux doit construire un environnement "*Niv_Ensemble*" contenant les deux classes. Il peut également affiner sa définition de l'historisation en précisant le filtre temporel de chaque classe.

```
Environment Niv_Ensemble { Hopital, Service };

Interface Hopital {
   D_attribute String nom;
   D_attribute String ville;
   D_attribute Float budget;
   C_attribute Short nb_services;
   S_attribute Short creation;
   D_composite dependent exclusive {hierarchy(Hopital)}
             relationship Set< Service > organisation
             delete restrict; }
with temporal filter {(budget, budget), (nb_services, nb_services),
                      (organisation, organisation)};

Interface Service {
   D_attribute String nom;
   D_attribute String téléphone;
   C_attribute Short nb_personnel;    }
with temporal filter {(nb_personnel, nb_personnel)};
```

4.4 Définition d'un entrepôt

Nous caractérisons un entrepôt de données par la définition suivante :

Définition :
Un **entrepôt** *ED* est défini par le n-uplet (Nom^{ED}, C^{ED}, Env^{ED}, $Config^{ED}$)

- Nom^{ED} est le nom de l'entrepôt,
- $C^{ED}=\{c_1, c_2,\ldots, c_n\}$ est l'ensemble fini des classes de l'entrepôt,
- $Env^{ED}=\{env_1, env_2,\ldots, env_m\}$ est l'ensemble fini des environnements de l'entrepôt,
- $Config^{ED}$ est l'ensemble des paramètres de configuration, visant à définir différents paramètres globaux.

Il est possible de définir des règles de configuration sur un entrepôt ; les règles globales s'appliquent à tout l'entrepôt, tandis que les règles locales à un environnement ne concernent que les données contenues dans cet environnement. Ces configurations globales déterminent le fonctionnement de l'entrepôt en terme de gestion des objets entrepôt ; nous n'abordons pas dans cette thèse les contraintes d'intégrités classiques dans les bases de données orientées objet [Amghar, Flory 1994]. L'administrateur peut définir une règle globale et la redéfinir

localement à un environnement ; pour cela, il donne le même nom que la règle globale à la règle locale (surcharge des règles).

EXEMPLE : Nous reprenons la définition de l'environnement "*Evolution*" de l'exemple de la section précédente. Il s'agit de définir les règles de configuration qui caractérisent le comportement temporel de l'entrepôt appelé "*MonED*" et de l'environnement "*Evolution*".

L'administrateur définit ainsi deux règles de configuration relatives à la période de rafraîchissement des données. Une règle globale définit une période de rafraîchissement annuelle pour l'entrepôt (appelé "*MonED*") et une règle locale à l'environnement "*Evolution*" définit une période de rafraîchissement mensuelle.

```
create configuration RAFRAICHIR on MonED
when T_Duree(now-self.Last_Refresh(), annee) > T_Duree(1, annee)
then self.Refresh();
create configuration RAFRAICHIR on Evolution
when T_Duree(now-self.Last_Refresh(), mois) > T_Duree(1, mois)
then self.Refresh();
```

4.5 Exemple complet

EXEMPLE : Nous décrivons un entrepôt de données (nommé *Professionnels_de_Sante*) qui contient des informations relatives à des personnes pouvant être des praticiens. Ces derniers travaillent dans des cabinets médicaux. Certains d'entre eux interviennent dans des services hospitaliers et parmi eux, quelques uns dirigent ces services. Les établissements hospitaliers sont composés de divers services. Les praticiens réalisent des consultations au cours desquelles ils prescrivent des médicaments.

L'entrepôt est défini par les classes entrepôt *Personne*, *Praticien*, *Cabinet*, *Visite*, *Etablissement* et *Service*. Elles se caractérisent de la manière suivante :

```
create warehouse Professionnels_de_sante;

interface Personne {
   D_attribute String nom;
   D_attribute String prenom;
   D_attribute String ville;
   }

interface Praticien extends Personne {
   D_attribute String specialite;
   D_attribute Short type_convention;
   D_relationship Cabinet travaille
                  inverse Cabinet::praticiens;
   D_relationship Set<Visite> consultations
                  inverse Visite::praticien;
   D_relationship Service intervient
                  inverse Service::equipe;
   D_relationship Service dirige
                  inverse Service::directeur;
   D_operation Boolean est_conventionne();
```

```
   D_operation String la_specialite();
   D_operation Double taux_teletransmission();
   D_operation Double ratio_generique();
   }
interface Cabinet {
   D_attribute String nom;
   D_attribute String ville;
   D_relationship Set<Praticien> praticiens
                 inverse Praticien::travaille;
   D_operation Short nb_praticiens();
   D_operation List<String> specialites();
   }
interface Visite {
   D_attribute String reference;
   D_attribute Double honoraire;
   D_attribut Boolean teletransmise;
   C_attribute Set<Struct> T_prescriptions {
      String code,
      Short categorie,
      Double tarif,
      Double taux_secu,
      Short type_molecule,
      Boolean generique } prescriptions;
   D_relationship Praticien praticien
                 inverse Praticien::consultations;
   D_relationship Patient patient
                 inverse Patient::consultations;
   D_operation Double VISITEtarif();
   D_operation Double montant_prescriptions();
   D_operation Boolean est_teletransmise();
   D_operation Double MEDICAMENTtarif();
   D_operation Boolean est_generique();
   }
interface Etablissement {
   D_attribute String nom;
   D_attribute Struct T_Adresse {
      String libelle,
      Short code,
      String ville } adresse;
   D_attribute String statut;
   D_attribute Struct T_subventions {
      Double montant,
      String institution,
      String type} subventions
   D_composite dependent exclusive {hierarchy(Etablissements)}
```

```
            relationship Set<Services> organisation
                         inverse Services::etablismt;
            delete restrict;
   D_operation Short nbServices();
   D_operation Boolean estPublic();
   D_operation Double budget();
   D_operation Double taxe();
   D_operation Double depart();
   D_operation Double region();
   }
with temporal filter {(subventions, subventions), (budget, budget())},
     archive filter {(montant, avg(subventions.montant)), (budget, avg(budget))}
            with T_Unit(annee) by 2;

interface Service {
   D_attribute String nom;
   D_attribute Double budget;
   D_relationship Set<Praticien> equipe
                inverse Praticien::intervient;
   D_relationship Praticien directeur
                inverse Praticien::dirige;
   D_relationship Etablissements etablismt
                inverse Etablissements::organisation;
   D_operation Short tailleEquipe();
   D_operation Boolean estPublic();
   D_operation Double budget();
   }
with temporal filter {(budget, budget)},
     archive filter {(budget, avg(budget))}
            with T_Unit(annee) by 2;
```

L'administrateur désire conserver des évolutions de valeurs concernant les établissements hospitaliers et leur organisation. Plus précisément, il veut conserver les évolutions détaillées du budget de chaque service et les subventions obtenues par les établissements hospitaliers. Cette conservation détaillée est actualisée tout les mois et elle est conservée durant 5 ans. Après ce délai, l'archivage est réalisé. Ainsi, les classes *Etablissement* et *Service* possèdent des filtres temporels et d'archives. Elles sont englobées dans un environnement, dont le comportement temporel est caractérisé par une règle de configuration.

```
environment SuiviHopitaux { Etablissement, Service };

create configuration ARCHIVAGE on SuiviHopitaux
when self.Refresh()
if select E
   from C in self.Classes(), O in C.Extension(), E in O.Histoire()
   where T_Duree(now-Debut(E.DomT()), annee) > T_Duree(5, annee)
then E.Archive(O);
```

```
create configuration RAFRAICHIR on Professionnel_de_sante
when T_Duree(now-self.Last_Refresh(), mois) > T_Duree(1, mois)
then self.Refresh();
```

4.6 Synthèse

Nous avons défini le concept d'environnement permettant de spécifier des parties historisées ayant un comportement temporel homogène. Le concept d'environnement unifie les granularités d'historisation en intégrant trois niveaux (attribut, classe et ensemble).

En plus de la généralisation des historisations (unification des granularités d'historisation), l'administrateur peut définir des règles de configuration de l'entrepôt et des environnements (concernant leur comportement temporel). Elles permettent de définir un entrepôt flexible qui s'adapte aux besoins de l'administrateur pour conserver uniquement les évolutions pertinentes.

5 CONCLUSION

Ce chapitre présente notre proposition de modélisation des entrepôts de données complexes et évolutives. Le modèle de données, que nous avons défini, apporte des solutions nouvelles répondant aux exigences des entrepôts (conservation des évolutions de manière adaptée aux besoins des applications décisionnelles). Ces contraintes n'existent pas dans les bases de données classiques et les bases de données temporelles. En particulier, nous avons défini trois concepts.

Le concept d'**objet entrepôt** étend le concept d'objet en intégrant l'aspect évolutif des données. Il se compose d'un état courant, de plusieurs états passés représentant les évolutions détaillées et d'états archivés représentant certaines évolutions de manière résumées. L'intérêt de cette proposition réside dans le fait que les évolutions des données sont conservées avec un niveau de détail pertinent. Les évolutions sont modélisées à partir d'un modèle temporel linéaire, multigranulaire et discret.

Le concept de **classe entrepôt** étend le concept standard de classe pour prendre en compte les caractéristiques d'historisation et d'archivage des objets entrepôt au travers de filtres temporels et de filtres d'archives. Le filtre temporel caractérise les propriétés temporelles d'une classe entrepôt dont les évolutions sont conservées de manière détaillées. Le filtre d'archives caractérise les propriétés archivées dont les évolutions sont résumées. Deux d'archivages sont possibles, soit un archive fort qui résume fortement les valeurs passés en perdant la trace de l'évolution, soit un archivage modéré qui résume les évolutions à un niveau de détail plus élevé.

Le concept d'**environnement** permet de définir simplement, dans le schéma de l'entrepôt, des parties temporelles homogènes, cohérentes, de taille adaptée aux besoins décisionnels et configurables par l'administrateur. Ce concept permet d'unifier les niveaux d'historisation en offrant trois niveaux de granularité : attribut, classe ou ensemble. En outre, il octroie une grande adaptabilité à l'entrepôt en permettant à l'administrateur de définir des environnements ayant un comportement temporel spécifique afin de conserver uniquement les évolutions pertinentes

La contribution de notre modèle est donc double. Premièrement, nous avons défini un mécanisme d'historisation pour conserver les évolutions des données et un mécanisme original d'archivage automatique et adaptable. Ceci permet de limiter au strict nécessaire le volume des données, notamment des données temporelles. Deuxièmement, nous avons proposé un concept nouveau (environnement) qui unifie les niveaux d'historisation offrant la possibilité d'historiser uniquement les données dont le passé est nécessaire.

Le chapitre suivant vise à étudier le mécanisme d'extraction mis en jeu pour élaborer l'entrepôt de données (modélisé par la fonction de construction $Mapping^c$ associée aux classes entrepôt). En effet, dans notre architecture, l'entrepôt est dérivé de la source globale à l'aide d'un processus d'extraction et de construction complexe : le contexte orienté objet nécessite d'aborder l'extraction des données et des structures mais également des opérations c'est à dire du comportement des données.

CHAPITRE III :
PROCESSUS D'ELABORATION D'ENTREPOTS PAR EXTRACTIONS

1 INTRODUCTION A L'ELABORATION D'ENTREPOTS

1.1 Objectif

Ce chapitre présente les principes mis en œuvre pour l'élaboration de l'entrepôt. Plus précisément, nous étudions le mécanisme d'extraction permettant de construire un entrepôt à partir d'une source globale.

Au chapitre II, nous avons défini un modèle conceptuel de représentation des données dédié aux entrepôts. Dans une même classe entrepôt sont regroupés les données (*Extension*^{c}), les structures de données (*Structure*^{c}) ainsi que les comportements des données (*Comportement*^{c}). Notre processus d'élaboration de l'entrepôt aborde deux aspects.

- L'aspect statique des données consiste en un mécanisme de définition des données et des structures des classes entrepôt. Cette définition est réalisée à partir des classes de la source globale en dérivant uniquement les données et les structures pertinentes pour les applications décisionnelles.
- L'aspect dynamique des données consiste en un mécanisme de définition des comportements des classes entrepôt. Ce mécanisme doit garantir que les opérations dérivées dans l'entrepôt utilisent uniquement les données présentes dans l'entrepôt.

1.2 Existant

En règle générale, la technique utilisée pour définir un entrepôt de données est la technique des vues matérialisées [Widom 1995] [Gupta, Mumick 1995], calculées sur des sources relationnelles ; de très nombreux travaux étudient cette approche (*cf.* chapitre I) parmi lesquels plusieurs traitent de l'extraction des données source pour l'élaboration de l'entrepôt [Labio, Garcia-Molina 1996] [Chawathe, et al 1994]. Ces travaux proposent des algorithmes pour maintenir des vues matérialisées en fonction des changements survenus dans les relations source. Plus récemment, dans [Cui, Widom 2000] les auteurs abordent les problèmes liés à la correspondance entre les données stockées dans l'entrepôt et les données source dont elles sont issues (*the tracing lineage problem*). Ces travaux se concentrent sur les aspects de détection des changements des données source [Labio, Garcia-Molina 1996] ainsi que sur la mise en correspondance des données de la source et de l'entrepôt [Cui, et al 2000]. Nos travaux se distinguent de ceux-ci puisque les travaux précités se situent dans un contexte d'entrepôts relationnels, n'abordant pas les problèmes d'extraction des comportements de données sous forme d'objets.

Notre modèle d'entrepôt est un modèle orienté objet. L'intégration du concept de vue dans les bases de données objet consiste généralement à considérer une vue comme un schéma de classes [Abiteboul, Bonner 1991] [Bellahsene 1996]. A notre connaissance, il existe peu de système de gestion de bases de données orientées objet intégrant le mécanisme de vue. Seuls deux prototypes existent : O2Views [Souza, et al 1994] [Souza 1995] et MultiView [Kuno, Rundensteiner 1996] [Rundensteiner, et al 1996]. Ces solutions permettent de définir des vues objet sur un schéma source objet. Cependant, ces propositions sont mal adaptées à notre

contexte d'entrepôt puisque leur définition des vues intègre un processus automatique d'extraction des hiérarchies de classes [Souza, et al 1994] [Kuno, Rundensteiner 1996]. Ces travaux se contentent de répercuter les hiérarchies existantes dans le schéma de la vue, rendant difficile la réorganisation du schéma de l'entrepôt afin de l'adapter aux besoins des applications décisionnelles. Dans [Lacroix, et al 1998] les auteurs proposent une première solution visant à réorganiser le schéma de la vue : ils introduisent des opérations de généralisation et de spécialisation pour restructurer la hiérarchie d'héritage des classes contenues dans le schéma de la vue. Cependant, le processus d'extraction induit la dérivation automatique d'une partie des hiérarchies, rendant difficile la réorganisation. En outre, ces travaux ne traitent pas de l'extraction des comportements.

1.3 Proposition

Nous avons spécifié une solution permettant de définir l'entrepôt de données en deux étapes successives.

- La définition de l'aspect statique des classes entrepôt ; il s'agit de spécifier, à partir de la source globale, les données pertinentes et les structures de ces données pour définir celles des classes entrepôt [Teste 2000].
- La définition de l'aspect dynamique des classes entrepôt ; il s'agit de définir le comportement des classes entrepôt en fonction de celui des classes source et des éléments (propriétés et opérations) présents dans l'entrepôt [Ravat, Teste, Zurfluh 2000a].

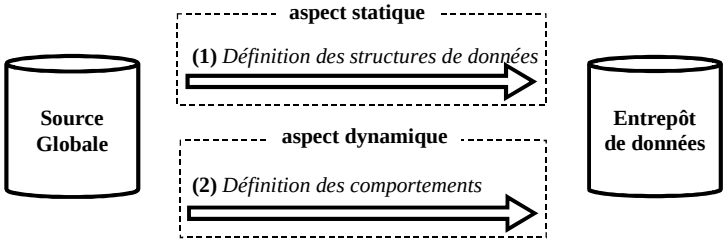


Figure 27 : Principe d'extraction pour l'élaboration de l'entrepôt.

Le processus de définition des structures et des données des classes entrepôt est réalisé au travers de la fonction de construction (pour une classe entrepôt *c*, on note $Mapping^c$ la fonction de construction). Cette fonction de construction est définie par une expression constituée d'une succession d'opérations de base.

La définition du comportement des classes entrepôt s'appuie sur des matrices d'usage. Notre approche consiste à déterminer de manière semi-automatique si les éléments (c'est à dire les attributs, les relations et les opérations) utilisés par une opération sont tous présents dans l'entrepôt.

Ce chapitre comporte deux sections.

- La section 2 présente notre approche pour spécifier les structures et les données source à partir desquelles sont construites celles de l'entrepôt ; il s'agit de définir la structure et l'extension des classes entrepôt.
- La section 3 présente notre solution pour définir le comportement des classes entrepôt à partir du comportement des classes source (impliquées dans la définition de l'aspect statique des classes entrepôt).

2 DEFINITION DE L'ASPECT STATIQUE DES CLASSES ENTREPOT

Le mécanisme de définition de l'aspect statique des classes entrepôt consiste à définir la structure des classes entrepôt ainsi que leur extension. Plus précisément, il s'agit pour l'administrateur de

- désigner les objets source qu'il souhaite recopier dans l'entrepôt,
- définir la structure des objets entrepôt créés.

La définition de l'aspect statique d'un classe entrepôt *c* repose sur sa fonction de construction $Mapping^c$. Cette fonction est formée d'une succession d'opérations de base parmi lesquelles nous distinguons les catégories suivantes :

- les **opérations de structuration** (FS) définissent la structure (attributs et relations) des classes entrepôt ;
- les **opérations de qualification** (FQ) déterminent les objets source à partir desquels l'extension de la classe entrepôt est calculée ; ces opérations offrent également des mécanismes pour combiner les objets afin de constituer des objets entrepôt adaptés aux besoins des décideurs ;
- les **opérations ensemblistes** (FE) issues des algèbres objet [Shaw, Zdonik 1990] ; elles offrent des mécanismes puissants pour transformer les classes afin de constituer des classes entrepôt adaptées aux besoins des décideurs ;
- les **opérations de hiérarchisation** (FH) organisent la hiérarchie d'héritage dans l'entrepôt en créant des super-classes et des sous-classes dans l'entrepôt. En effet, l'objectif de l'entrepôt étant différent de celui de la source globale, il est nécessaire de permettre la définition d'une nouvelle hiérarchie d'héritage entre les classes entrepôt.

La section 2.1 précise le principe d'extraction qui permet de recopier la valeur d'un objet source jugé (par l'administrateur) comme pertinent pour les décideurs dans un objet entrepôt qui le représente.

Les sections 2.2, 2.3, 2.4 et 2.5 définissent les opérations de définition de l'aspect statique des classes entrepôt.

La section 2.6 introduit un mécanisme permettant de conserver ou de ne pas conserver les hiérarchies (héritage et composition) existantes à la source globale.

La section 2.7 traite les ambiguïtés liées à la définition des associations et des compositions entre les classes entrepôt.

2.1 Principe de l'extraction des données

Une classe entrepôt *c* est définie par (Nom^c, $Type^c$, $Super^c$, $Extension^c$, $Mapping^c$, $Tempo^c$, $Archi^c$) où $Type^c$ est le type de la classe tel que $Type^c$=($Structure^c$, $Relation^c$, $Comportement^c$).

Un objet entrepôt *o* est défini par (*oid*, S_0, *Histoire*, *Archive*, *Origine*) où *oid* est son identifiant et

- S_0=($DomT_0$, V_0) est son état courant,

- *Histoire* est l'ensemble de ses états passés,
- *Archive* est l'ensemble de ses états archivés et
- *Origine* est l'ensemble des objets source dont il est issu.

L'extraction permet de recopier la valeur d'un objet source pertinent pour les décideurs dans un objet entrepôt. Une classe source *cs* est définie par (Nom^c, $Type^c$, $Super^c$, $Extension^c$). Afin de pouvoir appliquer une succession d'opérations pour extraire les données source, les objets source sont considérés comme des objets entrepôt particuliers *os*=(*s_oid*, S_0, *Histoire*, *Archive*, *Origine*) où *s_oid* est l'identifiant, S_0 est l'état courant contenant la valeur V_0 de l'objet, *Histoire*=∅, *Archive*=∅ et *Origine*={*os*}.

Les extractions sont effectuées à un point de synchronisation (instant pendant lequel les transactions de mise à jour des données sources sont toutes terminées), appelé un **point d'extraction**. A chaque point d'extraction, la valeur courante d'un objet entrepôt *o* est rafraîchie par la valeur courante de son origine source (on note cet état par le symbole ≙) :

- chaque attribut dérivé prend la valeur de l'attribut source qu'il représente,
- chaque attribut calculé prend la valeur retournée par le calcul appliqué sur la source,
- chaque relation dérivée relie les objets entrepôt représentant les objets source liés par la relation source.

Il est important de noter que le rafraîchissement des valeurs concerne les données dérivées et calculées. Les propriétés spécifiques des objets entrepôt (non extraites de la source globale) ne sont pas modifiées.

EXEMPLE : Soient un objet source modélisant une personne (identifié par sg_oid1) et un objet entrepôt (identifié par ed_oid1) issu de cet objet source tels que la valeur est rafraîchie $ed\_oid1.S_0.V_0 \triangleq sg\_oid1.S_0.V_0$ au point d'extraction t'.

- La valeur de chaque attribut dérivé (*D_nom*, *D_prenom*, *D_ville*) dans l'entrepôt est égale à la valeur de l'attribut source correspondant (*nom*, *prenom*, *adresse.ville*).
- La valeur de l'attribut calculé (*C_age*) est le résultat de l'opération calculant sa valeur *age*() appliquée sur l'objet source.
- La valeur de l'attribut spécifique (*S_taille*) n'est pas issue de l'origine source ; elle est fournie directement par les utilisateurs ou l'administrateur et elle est inchangée lors du rafraîchissement.

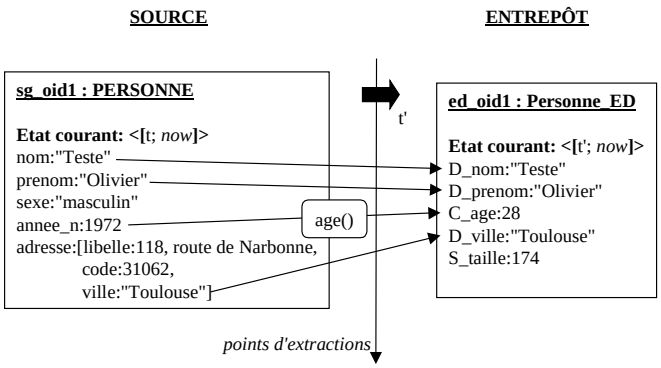


Figure 28 : Rafraîchissement de la valeur d'un objet entrepôt.

2.2 Opérations de structuration (FS)

Les opérations de structuration permettent de définir les attributs et les relations dérivés ainsi que les attributs calculés constituant la structure des classes entrepôt. On pose FS={π, μ, α} puisque les opérations de structuration comprennent :

- la projection (π),
- le masquage (μ) et
- l'accroissement (α).

2.2.1 Projection et masquage

La projection dérive un ensemble de propriétés de la classe source pour construire la structure de la classe entrepôt. Les propriétés de la classe source et de ses super-classes peuvent être dérivées pour construire la structure de la classe entrepôt.

<u>*Définition*</u> :
La **projection** π est définie par

π : $C^S \times 2^{PS} \rightarrow C^{ED}$

$(cs, \{p_1, p_2,\ldots,p_m\}) \rightarrow \pi(cs, \{p_1, p_2,\ldots,p_m\})=c \mid$

$AttributDerive^c=\{a \mid a\in \{p_1, p_2,\ldots,p_m\} \wedge (a\in Attribut^{cs} \vee (a\in Attribut^{cs'} \mid cs'\in Super^{cs}))\}$

$RelationDerive^c=\{r \mid r\in \{p_1, p_2,\ldots,p_m\} \wedge (r\in Relation^{cs} \vee (r\in Relation^{cs'} \mid cs'\in Super^{cs}))\}$

$Extension^c=\{ o \mid \exists os\in Extension^{cs}, o.S_0.V_0 \triangleq os.S_0.V_0\}$

Le masquage, inversement à la projection, dérive les propriétés de la classe source non contenues dans un ensemble de propriétés spécifiées $\{p_1, p_2,\ldots,p_m\}$. L'intérêt de cette opération, par rapport à la projection, est que la définition de la classe entrepôt reste valide malgré la suppression d'un attribut dérivé de la source.

<u>*Définition*</u> :
Le **masquage** μ est défini par

μ : $C^S \times 2^{PS} \rightarrow C^{ED}$

$(cs, \{p_1, p_2,\ldots,p_m\}) \rightarrow \pi(cs, \{p_1, p_2,\ldots,p_m\})=c \mid$

$AttributDerive^c=\{a \mid a\notin \{p_1, p_2,\ldots,p_m\} \wedge (a\in Attribut^{cs} \vee (a\in Attribut^{cs'} \mid cs'\in Super^{cs}))\}$

$RelationDerive^c=\{r \mid r\notin \{p_1, p_2,\ldots,p_m\} \wedge (r\in Relation^{cs} \vee (r\in Relation^{cs'} \mid cs'\in Super^{cs}))\}$

$Extension^c=\{ o \mid \exists os\in Extension^{cs}, o.S_0.V_0 \triangleq os.S_0.V_0\}$

<u>EXEMPLE</u> : Dans cet exemple, et dans les autres exemples de ce chapitre, nous utilisons la source globale décrite en annexe A.

Un administrateur désire extraire des informations relatives aux praticiens. Pour cela, il peut utiliser une projection appliquée à la classe source *PRATICIEN* en définissant les propriétés source qu'il juge pertinent de dériver pour générer la structure de la classe entrepôt. Chaque objet source de la classe *PRATICIEN* est dérivé et matérialisé au travers d'un objet entrepôt le représentant.

L'administrateur définit la fonction de construction (Mapping) au travers de l'expression suivante :

```
π(PRATICIEN, {nom, prenom, specialite})
```

L'expression suivante donne le masquage équivalent à la projection précédente.

```
μ(PRATICIEN, {adresse, naissance, num_prat, categorie, type_convention,
              diplomes, travaille, intervient, dirige, consultations} )
```

La définition suivante décrit le type de la classe entrepôt générée :

```
interface Praticien {
   D_attribute String nom;
   D_attribute String prenom;
   D_attribute String specialite;
   }
```

2.2.2 Accroissement

L'accroissement permet de créer de nouvelles propriétés qui sont, soit des attributs calculés, soit des propriétés spécifiques.

- Un attribut calculé correspond au résultat d'une fonction de calcul appliquée à la source ou à l'activation d'une fonction existante dans la source.
- Les propriétés spécifiques ne sont pas issues de la source (elles sont ajoutées par l'administrateur dans l'entrepôt afin de donner la possibilité aux utilisateurs de compléter l'information de l'entrepôt en fonction de leurs propres connaissances). Une propriété spécifique correspond à une nouvelle propriété ajoutée par l'administrateur.

On pose E l'ensemble des fonctions de calcul applicables sur la source et $Type^{ED}$ désigne l'ensemble des types définis dans l'entrepôt.

<u>*Définition*</u> :
L'**accroissement** α est défini par

α : $C^S \times 2^{PS} \times E \rightarrow C^{ED}$

$(cs, \{p_1{:}e_1, p_2{:}e_2,\ldots,p_m{:}e_m\}) \rightarrow \alpha(cs, \{p_1{:}e_1, p_2{:}e_2,\ldots,p_m{:}e_m\}) = c \mid$

$AttributCalcule^c = \{a_i \mid a_i \in \{p_1, p_2,\ldots,p_m\} \wedge (e_i \in E \vee e_i \in Comportement^{cs})\}$

$AttributSpecifique^c = \{a_i \mid a_i \in \{p_1, p_2,\ldots,p_m\} \wedge (e_i \in Type^{ED})\}$

$RelationSpecifique^c = \{r_i \mid r_i \in \{p_1, p_2,\ldots,p_m\} \wedge (e_i \in Type^{ED})\}$

$Extension^c = \{ o \mid \exists os \in Extension^{cs}, o.S_0.V_0 \triangleq os.S_0.V_0\}$

<u>*EXEMPLE*</u> *:* A partir de l'exemple précédent concernant les praticiens, l'administrateur compléte les propriétés dérivées en ajoutant :

-un attribut calculé correspondant au nombre de consultations que les praticiens effectuent

-un attribut spécifique représentant la nationalité des praticiens.

Pour cela, l'administrateur applique une projection, suivie d'un accroissement sur la classe source *PRATICIEN*.

```
π(α(PRATICIEN, {nb_consult:count(PRATICIEN.consultations),
                nationalite:String}),
  {nom, prenom, specialite})
```

L'expression suivante décrit le type de la classe entrepôt générée :

```
interface Praticien {
   D_attribute String nom;
   D_attribute String prenom;
   D_attribute String specialite;
   C_attribute Short nb_consult;
   S_attribute String nationalite;
   }
```

2.3 Opérations de qualification (FQ)

Les opérations de qualification permettent de déterminer l'extension des classes entrepôt en appliquant des opérations algébriques sur la source. Elles définissent des critères qualifiant les objets source pertinents à dériver dans l'entrepôt. Elles permettent également de regrouper ou de dégrouper des objets source pour reconstruire dans l'entrepôt des objets entrepôt. On pose FQ={σ, $\bowtie$, η, η^{-1}} puisque les opérations de qualification englobent :

- la sélection (σ),
- la jointure ($\bowtie$),
- le groupement (η) équivalent à l'opérateur "*nest*" et
- le dégroupement (δ)équivalent à l'opérateur "*unnest*".

2.3.1 Sélection

La sélection génère une classe entrepôt dont l'extension est calculée à partir d'une restriction de l'extension source exprimée au travers d'un prédicat de sélection *pred*. La structure de la classe entrepôt est dérivée de celle de la classe source et de ses super-classes.

On pose *PRED* l'ensemble des prédicats valides, C^S l'ensemble des classes source et C^{ED} l'ensemble des classes entrepôt.

<u>*Définition*</u> :
La **sélection** σ élabore une classe entrepôt *c* à partir d'une classe source *cs* conformément à la définition suivante :

π : $C^S \times PRED \rightarrow C^{ED}$

$(cs, pred) \rightarrow \pi(cs, pred) = c \mid$

$AttributDerive^c = \{a \mid a \in Attribut^{cs} \vee (a \in Attribut^{cs'} \mid cs' \in Super^{cs})\}$

$RelationDerive^c = \{r \mid r \in Relation^{cs} \vee (r \in Relation^{cs'} \mid cs' \in Super^{cs})\}$

$Extension^c = \{ o \mid \exists os \in Extension^{cs}, pred(os) \wedge o.S_0.V_0 \triangleq os.S_0.V_0\}$

<u>*EXEMPLE*</u> *:* L'administrateur de l'entrepôt élabore une classe entrepôt relative aux généralistes en effectuant une sélection sur la classe source *PRATICIEN*. Dans l'expression de la sélection, le nom

de la classe source *PRATICIEN* est associé à une variable *p* afin de faciliter l'expression du prédicat de sélection.

```
σ(p PRATICIEN, p.categorie="generaliste")
```

2.3.2 Jointure

La jointure génère une classe entrepôt à partir du produit cartésien entre deux classes source cs_1 et cs_2. Les classes source cs_1 et cs_2 se définissent par (Nom^{cs1}, $Type^{cs1}$, $Super^{cs1}$, $Extension^{cs1}$) et (Nom^{cs2}, $Type^{cs2}$, $Super^{cs2}$, $Extension^{cs2}$).

<u>*Définition*</u> :
La **jointure** $\bowtie$ est définie par

$\bowtie$: $C^S \text{ x } C^S \text{ x } PRED \rightarrow C^{ED}$

$(cs_1, cs_2, pred\_j) \rightarrow \pi(cs_1, cs_2, pred\_j)=c \mid$

$AttributDerive^c=\{a \mid a\in Attribut^{cs1} \vee (a\in Attribut^{cs'} \mid cs'\in Super^{cs1}) \vee a\in Attribut^{cs2} \vee (a\in Attribut^{cs''} \mid cs''\in Super^{cs2})\}$

$RelationDerive^c=\{r \mid r\in Relation^{cs} \vee (r\in Relation^{cs'} \mid cs'\in Super^{cs}) \vee r\in Relation^{cs2} \vee (r\in Relation^{cs''} \mid cs''\in Super^{cs2})\}$

$Extension^c=\{o \mid \exists os_1\in Extension^{cs1}, \exists os_2\in Extension^{cs2}, pred\_j(os_1, os_2) \wedge o.S_0.V_0 \triangleq union(os_1.S_0.V_0, os_2.S_0.V_0)\}$

<u>*EXEMPLE*</u> *:* L'administrateur désire construire une classe entrepôt contenant les informations sur les praticiens généralistes et leur cabinet médical.

```
⋈(p σ(pp PRATICIEN, pp.categorie="generaliste"),
   c CABINET,
   p.travaille=c)
```

2.3.3 Groupement

L'opération de groupement crée une classe entrepôt dont les objets sont issus d'un regroupement de plusieurs objets source. Soient *PS* l'ensemble des propriétés des classes source et *ATT* l'ensemble des noms valides d'attributs.

<u>*Définition*</u> :
Le **groupement** η se définit par

η : $C^S \text{ x } 2^{PS} \text{ x } ATT \rightarrow C^{ED}$

$(cs, \{a_1, a_2,\ldots,a_m\}, att) \rightarrow \pi(cs, \{a_1, a_2,\ldots,a_m\}, att)=c \mid$

$Attribut^c=\{a \mid (a\in Attribut^{cs} \vee (a\in Attribut^{cs'} \mid cs'\in Super^{cs}))\}\cup\{attr\}$

$RelationDerive^c=\{r \mid r\in Relation^{cs} \vee (r\in Relation^{cs'} \mid cs'\in Super^{cs})\}$

$Extension^c=\{o \mid \exists os\in Extension^{cs}, (o.S_0.V_0=[p_1{:}os.S_0.p_1,\ldots, attr{:}t,\ldots, p_p{:}os.S_0.p_p] \mid \exists r\in t, os.S_0.attr=r \wedge \forall i\in[1..p], p_i\in Attribut^c\cup Relation^c)\}$

<u>*EXEMPLE*</u> *:* L'administrateur souhaite réorganiser les objets de la classe *PERSONNE* en les regroupant par ville de résidence.

η(PERSONNE, {adresse.ville}, les_personnes)

Les objets entrepôt regroupent les personnes résidant dans la même ville. La structure de la classe entrepôt produite est définie par l'expression suivante :

```
interface Personne {
   D_attribute String ville;
   D_attribute Set(Struct T_les_personnes {
      String nom,
      String prenom,
      Struct T_Adresse {
         String libelle, Short code, String ville } adresse,
      Struct T_Date {
         Short annee, Short mois, Short jour } naissance
      }) les_personnes;
   }
```

2.3.4 Dégroupement

L'opération de dégroupement consiste à décomposer un objet source en plusieurs objets entrepôt. Pour chaque valeur d'un attribut multivalué des objets source, un objet entrepôt est produit (dans le cas particulier d'un attribut monovalué, chaque objet source donne lieu à un seul objet entrepôt).

Définition :
Le **dégroupement** η^{-1} est défini par

δ : $C^S \times PS \rightarrow C^{ED}$

$(cs, att) \rightarrow \pi(cs, att) = c \mid$

$Attribut^c = \{a \mid a \in Attribut^{cs} \vee (a \in Attribut^{cs'} \mid cs' \in Super^{cs})\}$

$Relation^c = \{r \mid r \in Relation^{cs} \vee (r \in Relation^{cs'} \mid cs' \in Super^{cs})\}$

$Extension^c = \{o \mid \exists os \in Extension^{cs}, (o.S_0.V_0 = [p_1{:}os.S_0.p_1, \ldots, attr{:}t, \ldots, p_p{:}os.S_0.p_p] \mid t \in os.S_0.attr \wedge \forall i \in [1..p], p_i \in Attribut^c \cup Relation^c)\}$

EXEMPLE : L'administrateur applique l'opération de dégroupement sur la classe *Personne* obtenue précédemment (il ne s'agit pas d'une opération inverse).

δ(Personne, les_Personnes)

La structure de la classe entrepôt produite est définie par l'expression suivante :

```
interface PersonneDeg {
   D_attribute String ville;
   D_attribute String nom;
   D_attribute prenom;
   D_attribute T_Adresse {
         String libelle, Short code, String ville } adresse;
   D_attribute T_Date {
         Short annee, Short mois, Short jour } naissance;
   }
```

2.4 Opérations ensemblistes (FE)

Les opérations ensemblistes permettent de combiner plusieurs classes afin de définir une nouvelle classe entrepôt. Elles regroupent les opérations traditionnelles des algèbres objet, c'est à dire l'union ($\cup$), l'intersection ($\cap$) et la différence ($-$).

En orienté objet, deux types d'union, de différence et d'intersection sont définis, l'une basée sur l'égalité de valeur (=), l'autre basée sur l'égalité d'identifiant ($=_I$). On pose FE={$\cup$, $\cap$, $-$, $\cup^V$, $\cap^V$, $-^V$ }.

2.4.1 Union

L'opération d'union entre deux classes cs_1 et cs_2 de même type crée une classe entrepôt dont l'extension est obtenue à partir des objets source contenus dans les extensions de cs_1 ou de cs_2.

<u>*Définition*</u> :
L'**union** $\cup$ est définie par

$\cup$: $C^S \text{ x } C^S \rightarrow C^{ED}$

$(cs_1, cs_2) \rightarrow \cup(cs_1, cs_2)=c \mid$

$Attribut^c=\{a \mid a \in Attribut^{cs1} \vee (a \in Attribut^{cs'} \mid cs' \in Super^{cs1})\}$

$Relation^c=\{r \mid r \in Relation^{cs1} \vee (r \in Relation^{cs'} \mid cs' \in Super^{cs1})\}$

Si $\cup(cs_1, cs_2)$ alors
$Extension^c=\{o \mid \exists os \in \bigcup_{i \in \{1,2\}} Extension^{csi}, o.S_0.V_0 \triangleq os.S_0.V_0 \wedge \neg(\exists o' \in Extension^c \mid o=_I o')\}$,

Si $\cup^V(cs_1, cs_2)$ alors
$Extension^c=\{o \mid \exists os \in \bigcup_{i \in \{1,2\}} Extension^{csi}, o.S_0.V_0 \triangleq os.S_0.V_0 \wedge \neg(\exists o' \in Extension^c \mid o=o')\}$.

2.4.2 Intersection

L'opération d'intersection entre deux classes cs_1 et cs_2 de même type crée une classe entrepôt dont l'extension est obtenue à partir des objets source contenus dans les extensions de cs_1 et cs_2.

<u>*Définition*</u> :
L'**intersection** $\cap$ est définie par

$\cap$: $C^S \text{ x } C^S \rightarrow C^{ED}$

$(cs_1, cs_2) \rightarrow \cap(cs_1, cs_2)=c \mid$

$Attribut^c=\{a \mid a \in Attribut^{cs1} \vee (a \in Attribut^{cs'} \mid cs' \in Super^{cs1})\}$

$Relation^c=\{r \mid r \in Relation^{cs1} \vee (r \in Relation^{cs'} \mid cs' \in Super^{cs1})\}$

Si $\cap(cs_1, cs_2)$ alors
$Extension^c=\{o \mid (\exists os_1 \in Extension^{cs1}, o.S_0.V_0 \triangleq os_1.S_0.V_0) \wedge (\exists os_2 \in Extension^{cs2} \mid os_1 =_I os_2)\}$,

Si $\cap^V(cs_1, cs_2)$ alors
$Extension^c=\{o \mid (\exists os_1 \in Extension^{cs1}, o.S_0.V_0 \triangleq os_1.S_0.V_0) \wedge (\exists os_2 \in Extension^{cs2} \mid os_1 = os_2)\}$.

2.4.3 Différence

L'opération de différence entre deux classes cs_1 et cs_2 de même type crée une classe entrepôt dont l'extension est obtenue à partir des objets source contenus dans l'extension de cs_1 et non contenus dans l'extension de cs_2.

<u>*Définition*</u> :
La **différence** – est définie par

$-: \quad C^S \text{ x } C^S \rightarrow C^{ED}$

$(cs_1, cs_2) \rightarrow -(cs_1, cs_2) = c \mid$

$Attribut^c = \{a \mid a \in Attribut^{cs1} \vee (a \in Attribut^{cs'} \mid cs' \in Super^{cs1})\}$

$Relation^c = \{r \mid r \in Relation^{cs1} \vee (r \in Relation^{cs'} \mid cs' \in Super^{cs1})\}$

Si $-(cs_1, cs_2)$ alors
$Extension^c = \{o \mid (\exists os_1 \in Extension^{cs1}, o.S_0.V_0 \triangleq os_1.S_0.V_0) \wedge \neg(\exists os_2 \in Extension^{cs2} \mid os_1 =_1 os_2)\}$,

Si $-V(cs_1, cs_2)$ alors
$Extension^c = \{o \mid (\exists os_1 \in Extension^{cs1}, o.S_0.V_0 \triangleq os_1.S_0.V_0) \wedge \neg(\exists os_2 \in Extension^{cs2} \mid os_1 = os_2)\}$.

2.5 Opérations de hiérarchisation (FH)

L'entrepôt de données et la source globale sont deux systèmes autonomes dont les objectifs ne sont pas identiques. L'entrepôt de données doit être organisé de manière adaptée pour la gestion de l'information décisionnelle. Sa hiérarchie d'héritage entre les classes entrepôt est généralement différente de celle de la source globale.

Nous introduisons deux opérations pour généraliser et spécialiser les classes entrepôt afin de construire une hiérarchie d'héritage adaptée aux exigences des décideurs. On pose FH={Λ, Σ} l'ensemble des opérations de hiérarchisation ; elles s'appliquent sur les classes de l'entrepôt et comprennent

- la généralisation (Λ) et
- la spécialisation (Σ).

2.5.1 Généralisation

La généralisation génère une super-classe entrepôt à partir d'une ou plusieurs classes entrepôt, en regroupant l'ensemble des propriétés communes.

<u>*Définition*</u> :
La **généralisation** Λ est définie par

$\Lambda: \quad C^{ED} \text{ x} \ldots \text{x } C^{ED} \rightarrow C^{ED}$

$(c_1, \ldots, c_n) \rightarrow \Lambda(c_1, \ldots, c_n) = c \mid$

$Attribut^c = \{a \mid a \in \bigcap_{i=1}^{n} Attribut^{ci}\}$, $Relation^c = \{r \mid r \in \bigcap_{i=1}^{n} Relation^{ci}\}$, $Super^c = \{c \mid c \in \bigcap_{i=1}^{n} Super^{ci}\}$

$Extension^c = \{o \mid \exists o' \in \bigcup_{i=1}^{n} Extension^{ci}, o.S_0.V_0 \triangleq o'.S_0.V_0\}$

L'opération de généralisation crée une nouvelle super-classe dans l'entrepôt, mais elle provoque également des modifications dans la définition des classes entrepôt initiales.

$$\forall k \in [1..n],\ Attribut^{ck}=\{a \mid a \notin \bigcap_{i=1}^{n} Attribut^{ci} \ \wedge a \in Attribut^{ck}\}$$

$$Relation^{ck}=\{r \mid r \notin \bigcap_{i=1}^{n} Relation^{ci} \ \wedge r \in Relation^{ck}\}$$

$$Super^{ck}=\{c \mid c \in Super^{ck} \wedge c \notin \bigcup_{i=1, i \neq k} Super^{ci}\} \cup \{c\}$$

EXEMPLE : Soient deux classes entrepôt, *Specialiste* et *Generaliste*, définies par les fonctions de construction suivantes :

```
π(p σ(pp PRATICIEN, pp.categorie="specialiste"),
  {p.nom, p.prenom, p.categorie, p.specialite})
```

```
π(p σ(pp PRATICIEN, pp.categorie="generaliste"),
  {p.nom, p.prenom, p.categorie })
```

L'administrateur souhaite créer une super-classe représentant les personnes. Il utilise la fonction de généralisation suivante :

```
Λ(Specialiste, Generaliste)
```

Les expressions suivantes décrivent les types des classes générées :

```
interface Personne {
   D_attribute String nom;
   D_attribute String prenom;
   D_attribute String categorie;
   }

interface Specialiste extends Personne {
   D_attribute String specialite;
   }

interface Generaliste extends Personne {
   }
```

2.5.2 Spécialisation

La spécialisation génère une sous-classe entrepôt à partir d'une ou plusieurs classes entrepôt.

Définition :
La **spécialisation** Σ est définie par

$$\Sigma : \quad C^{ED} \times \ldots \times C^{ED} \times PRED \rightarrow C^{ED}$$

$$(c_1,\ldots, c_n, pred\_j) \rightarrow \Sigma(c_1,,\ldots, c_n, pred\_j)=c \mid$$

$$Super^{c}=\{c_1, c_2,\ldots,c_n\}$$

$$Extension^{c}=\{o \mid \exists o_1 \in Extension^{c1},\ldots,\exists o_n \in Extension^{cn}, pred\_j(o_1,\ldots, o_n) \wedge o.S_0.V_0 \triangleq [o_1.S_0.V_0,\ldots, o_n.S_0.V_0]\}$$

Remarque :
La spécialisation couplée avec une sélection permet de restreindre l'ensemble des objets placés dans la sous-classe générée.

2.6 Traitement des hiérarchies existantes

L'entrepôt de données et la source de données sont deux systèmes autonomes aux objectifs différents : la source globale contient l'ensemble des informations issues de l'intégration de sources distribuées, autonomes et hétérogènes tandis que l'entrepôt de données centralise les informations utiles pour les décideurs. Comme nous l'avons introduit dans la section précédente, ces divergences se répercutent par des différences entre la hiérarchie d'héritage de l'entrepôt et la hiérarchie d'héritage de la source globale.

Cependant, il est fastidieux et même peu satisfaisant pour l'administrateur de reconstruire une hiérarchie existante. Toutes les opérations de structuration, de qualification et ensemblistes sont étendue pour indiquer :

- si la hiérarchie d'héritage des sous-classes à une classe source extraite est conservée ou non (par défaut, elle ne l'est pas),
- si la hiérarchie de composition des classes composantes une classe source extraite est conservée ou non (par défaut, elle ne l'est pas).

Définition :
Nous introduisons les extensions H et C telles que

$\forall f \in FS \cup FQ \cup FE$, si $f^H(cs)$ alors la hiérarchie d'héritage de *cs* est conservée
si $f(cs)$ alors la hiérarchie d'héritage de *cs* n'est pas conservée.

$\forall f \in FS \cup FQ \cup FE$, si $f^C(cs)$ alors la hiérarchie de composition de *cs* est conservée
si $f(cs)$ alors la hiérarchie de composition de *cs* n'est pas conservée.

EXEMPLE : Soient les deux fonctions de construction suivantes :

```
πH(p σ(pp PRATICIEN, pp.categorie="specialiste"),
  {p.nom, p.prenom, p.adresse, p.categorie, p.specialite })
πH(p σ(pp PRATICIEN, pp.categorie="generaliste"),
  {p.nom, p.prenom, p.categorie })
```

La hiérarchie d'héritage source entre *PRATICIEN* et *PERSONNE* est reconstruite automatiquement dans l'entrepôt. Cependant, la première fonction projette dans l'entrepôt trois attributs de la classe *PERSONNE* tandis que la seconde fonction n'en projette que deux : l'attribut *adresse* n'est pas commun aux deux fonctions. Plusieurs classes entrepôt sont produites et elles correspondent aux types suivants :

```
interface PERSONNE1 {
   D_attribute String nom;
   D_attribute String prenom; }

interface PERSONNE2 extends PERSONNE1 {
   D_attribute Struct T_Adresse {
      String libelle,
```

```
      Short code,
      String ville } adresse; }

interface PRATICIEN1 extends PERSONNE1 {
   D_attribute String categorie; }

interface PRATICIEN2 extends PERSONNE2 {
   D_attribute String categorie; }

interface Specialiste extends PRATICIEN2 {
   D_attribute String specialite; }

interface Generaliste extends PRATICIEN1 { }
```

2.7 Traitement des ambiguïtés de définition des relations d'association et de composition

Notre approche consiste à définir les structures des classes entrepôt à partir des structures de la source globale. Lors de la dérivation dans l'entrepôt d'une relation d'association ou de composition entre des classes source, cette relation est redéfinie entre des classes de l'entrepôt.

EXEMPLE : Considérons la classe entrepôt *Praticien* obtenue par la fonction de construction suivante :

π(PRATICIEN, {nom, prenom, categorie, specialite, intervient})

La relation *intervient* inverse de *equipe* relie à la source la classe *PRATICIEN* à *SERVICE*. Ceci oblige l'administrateur à définir au moins une classe entrepôt représentant les services.

π(SERVICE, {nom, equipe})

Ainsi, le processus d'élaboration par extractions de l'entrepôt reconstruit les relations entre les praticiens et les services dans lesquels ils travaillent conformément à la source.

Cette approche permet de maintenir les deux systèmes (entrepôt et source globale) indépendants et autonomes. Cependant un problème apparaît lorsque plusieurs classes entrepôt représentent une même classe source participant à une relation.

EXEMPLE : Considérons deux classes entrepôt *ServicesHospitaliers* et *ServicesCliniques* issues de la classe source *SERVICE* selon les fonctions de construction suivantes :

π(σ(s SERVICE, s.etablissmt.statut="public"), {nom, budget})

π(σ(s SERVICE, s.etablissmt.statut="prive"), {nom})

Si l'administrateur définit une autre classe entrepôt *Etablissements* comportant la relation de composition *organisation*, une ambiguïté survient.

π(ETABLISSEMENT, {nom, statut, organisation})

La relation de composition *organisation* doit être redéfinie entre les classes entrepôt. Elle relie la classe entrepôt *Etablissements* avec une (ou plusieurs) classe(s) entrepôt représentant la classe source *SERVICE* (classe impliquée dans la relation). Or, deux classes entrepôt, *ServicesHospitaliers* et *ServicesCliniques* sont issues de *SERVICE*.

Nous proposons un mécanisme permettant à l'administrateur de spécifier les classes entrepôt reliées lorsqu'une ambiguïté existe. Nous introduisons la notation particulière *rel|classe_liée* permettant d'indiquer qu'une relation sémantique *rel* est redéfinie avec la classe entrepôt *classe_liée* représentant la classe source reliée.

> *EXEMPLE :* Nous reconsidérons la définition de la classe entrepôt *Etablissements* :
>
> π(ETABLISSEMENT, {nom, statut, organisation|ServiceHospitaliers})
>
> La relation *organisation* dérivée dans l'entrepôt relie la classe entrepôt *Etablissements* à la seule classe entrepôt *ServicesHospitaliers*.

Si l'administrateur ne spécifie pas une classe entrepôt liée lors d'ambiguïté, le système relie toutes les classes candidates.

2.8 Exemple complet

Dans cette section, nous décrivons l'exemple complet de construction de l'entrepôt de données à partir de la source globale (décrite en annexe A). Il s'agit des fonctions de construction relatives aux structures de l'entrepôt décrites dans l'exemple complet du chapitre II.

> *EXEMPLE :* Nous reprenons l'exemple de l'entrepôt défini à la section 4.5 du chapitre II. Les classes de l'entrepôt sont obtenues à partir d'une source globale (décrite en annexe A) en appliquant les fonctions de construction suivantes :
>
> -Extraction des établissements.
>
> ```
> π(e ETABLISSEMENT,
>  {e.nom, e.adresse, e.statut, e.subvention, e.organisation})
> ```
>
> -Extraction des services qui composent les établissements.
>
> ```
> π(s SERVICE,
>  {s.nom, s.budget, s.etablissmt, s.equipe, s.directeur})
> ```
>
> -Extraction des spécialistes en conservant la hiérarchie d'héritage source.
>
> ```
> μ^H(p σ^H(pp PRATICIEN, pp.categorie="specialiste"),
>  {p.naissance, p.categorie, p.adresse.libelle, p.adresse.code,
>  p.diplôme )
> ```
>
> -Extraction des cabinets médicaux des spécialistes.
>
> ```
> π(c ⋈(cc CABINET,
>  pp σ(pr PRATICIEN, pr.categorie="specialiste"),
>  pp∈cc.praticiens),
>  {c.nom, c.CABINETadresse.ville, c.praticiens})
> ```
>
> -Extraction des visites effectuées par les patients chez des spécialistes ainsi que des prescriptions ordonnées à cette occasion par les professionnels de santé.
>
> ```
> η(g π(v ⋈(pr ⋈(vv VISITE,
>  pp σ(p PRATICIEN, p.categorie="specialiste"),
>  vv∈pp.consultations),
>  me MEDICAMENT,
>  me∈pr.prescription),
> ```

```
      {v.reference, v.honoraire, v.teletransmise, v.praticien,
       v.code, v.categorie, v.tarif, v.taux_secu, v.type_molecule,
       v.generique}),
  {g.reference, g.honoraire, g.teletransmise, g.praticien},
  prescriptions)
```

2.9 Synthèse

Cette section traite de la définition de l'aspect statique (structure et extension) des classes entrepôt. Notre solution fournit un ensemble d'opérations issues des algèbres objet permettant de définir les classes entrepôt. Le tableau suivant résume l'ensemble des opérations.

CATEGORIES	OPERATIONS	DESCRIPTIONS	
Qualification	***Sélection***	*Syntaxe :*	$\sigma(cs, pred)=c$
		Rôle :	Restreindre l'extension de *cs* à partir de laquelle est peuplée celle de *c*.
	Jointure	*Syntaxe :*	$\bowtie(cs_1, cs_2, pred\_j)=c$
		Rôle :	Filtrer le produit cartésien entre cs_1 et cs_2 à partir duquel est peuplée l'extension de *c*.
	Groupement	*Syntaxe :*	$\eta(cs, \{p_1, p_2,\ldots,p_m\}, attr)=c$
		Rôle :	Regrouper des objets de *cs* pour peupler *c*.
	Dégroupement	*Syntaxe :*	$\delta(cs, attr)=c$
		Rôle :	Décomposer des objets multivalués de *cs* pour peupler *c*.
Ensembliste	***Union***	*Syntaxe :*	$\cup(cs_1, cs_2)=c$ *ou* $\cup^{V}(cs_1, cs_2)=c$
		Rôle :	Réaliser l'union de cs_1 et cs_2 pour générer *c*.
	Intersection	*Syntaxe :*	$\cap(cs_1, cs_2)=c$ *ou* $\cap^{V}(cs_1, cs_2)=c$
		Rôle :	Réaliser l'intersection de cs_1 et cs_2 pour générer *c*.
	Différence	*Syntaxe :*	$-(cs_1, cs_2)=c$ *ou* $-^{V}(cs_1, cs_2)=c$
		Rôle :	Réaliser la différence entre cs_1 et cs_2 pour générer *c*.
Structuration	***Projection***	*Syntaxe :*	$\pi(cs, \{p_1, p_2,\ldots,p_m\})=c$
		Rôle :	Définir les propriétés de *cs* qui sont extraites pour élaborer la structure de *c*.
	Masquage	*Syntaxe :*	$\mu(cs, \{p_1, p_2,\ldots,p_m\})=c$
		Rôle :	Définir les propriétés de *cs* qui ne sont pas extraites pour élaborer la structure de *c*.
	Accroissement	*Syntaxe :*	$\alpha(cs, \{p_1{:}e_1, p_2{:}e_2,\ldots,p_m{:}e_m\})=c$
		Rôle :	Définir les propriétés supplémentaires qui sont ajoutées à la structure de *c* élaborée avec l'ensemble des propriétés de *cs*.
Hiérarchisation	***Généralisation***	*Syntaxe :*	$\Lambda(c_1, c_2,\ldots, c_n)=c$
		Rôle :	Créer une super classe à $c_1, c_2,\ldots, c_n$ dont le type est constitué des propriétés communes.
	Spécialisation	*Syntaxe :*	$\Sigma(c_1, c_2,\ldots, c_n, pred\_j)=c$
		Rôle :	Créer une sous classe à $c_1, c_2,\ldots, c_n$.

Tableau 11 : Opérations de base pour exprimer la fonction de construction (*Mapping*^{c}).

L'intérêt de notre approche est de permettre à l'administrateur de sélectionner les informations utiles pour les décideurs. Notre approche s'appuie sur la puissance des langages algébriques orientés objet.

Puisque les besoins des applications décisionnelles (OLAP) diffèrent de ceux des applications de la source (OLTP) [Codd 1993] [Inmon 1994] [Kimball 1996], les classes entrepôt sont organisées de manière différente par rapport à la source globale. Nous proposons des mécanismes suffisamment flexibles pour restructurer les données et refondre la hiérarchie d'héritage suivants les besoins des applications décisionnelles. En outre, nous avons proposé une solution permettant à l'administrateur de choisir entre la conservation ou non des hiérarchies d'héritage et de composition de la source globale.

Dans notre contexte orienté objet, il reste à traiter de la définition du comportement des classes entrepôt.

3 DEFINITION DE L'ASPECT DYNAMIQUE DES CLASSES ENTREPOT

Le paradigme objet que nous adoptons, regroupe dans une même entité structure et comportement. Après avoir étudié la définition des structures de données, cette section traite de la définition du comportement de ces données. Plus précisément, l'objectif de cette section est de proposer une solution visant à définir semi-automatiquement les comportements des classes entrepôt à partir de leur fonction de construction ($Mapping^c$).

Nous proposons de dériver, dans l'entrepôt de données, les opérations définies dans la source globale. Ceci est réalisé en se basant sur un processus semi-automatique,

- qui détermine si les attributs et les relations nécessaires à une opération sont présentes dans l'entrepôt,
- qui détermine si les opérations invoquées par une opération sont présentes dans l'entrepôt.

A notre connaissance, aucune proposition ne traite de ce problème dans le domaine des entrepôts ; la raison principale étant que les entrepôts de données sont habituellement développés dans un contexte relationnel où seules les structures sont dérivées par le biais de la technique des vues matérialisées [Widom 1995] [Gupta, Mumick 1995].

3.1 Opération dérivable

La fonction de construction $Mapping^c$ (la fonction de construction pour une classe entrepôt *c*) met en jeu différentes classes source pour créer la structure et l'extension d'une classe entrepôt. Il est alors nécessaire de définir également le comportement de cette classe.

Pour dériver les opérations d'une classe source, nous devons déterminer si l'ensemble des propriétés et des opérations requises par chaque opération est présent dans l'entrepôt. Nous caractérisons une opération comme étant dérivable si elle satisfait à la définition suivante :

Définition :
Une opération est une **opération dérivable** dans une classe entrepôt si et seulement si

- toutes les propriétés (attributs et relations) utilisées par l'opération sont présentes dans la classe entrepôt,
- toutes les opérations invoquées par l'opération sont présentes dans l'entrepôt.

Une opération op_{a1} manipule uniquement les propriétés (attributs et relations) locales à la classe tandis qu'une opération op_{a2} d'une autre classe peut être invoquée par l'envoie d'un message. Autrement dit, les opérations manipulent les propriétés locales ainsi que les opérations locales à la classe et les opérations des autres classes (par l'envoie de messages).

Par conséquent, le processus de définition des comportements s'appuie sur une analyse à deux niveaux :

- **localement** à chaque classe entrepôt générée pour déterminer si toutes les propriétés nécessaires aux opérations sont disponibles,
- **globalement** pour déterminer si toutes les opérations invoquées par les opérations sont disponibles.

3.2 Matrices d'usage

Pour déterminer la "*dérivabilité*" des opérations, la solution que nous proposons s'inspire de la technique des matrices d'usage utilisées dans la conception des bases de données réparties [Ravat, Zurfluh 1995]. Nous étendons le concept de matrice d'usage proposé dans les bases de données réparties afin de l'adapter au contexte de la conception des entrepôts objet.

Dans notre contexte, une matrice d'usage contient les différents critères à analyser à partir desquels la décision de dériver une opération est effectué. Une matrice d'usage est construite de la manière suivante.

- Les opérations éventuellement dérivables forment les lignes et les critères à analyser constituent les colonnes de la matrice . Pour chaque opération, la valeur 1 indique les critères nécessaires à l'opération.
- Une ligne supplémentaire, nommée "*Dérivé*", précise les critères présents (ou dérivés) dans l'entrepôt.
- Une colonne supplémentaire, nommée "*Dérivable*", permet de décider de la "*dérivabilité*" ou non de chaque opération en fonction de la présence dans l'entrepôt des critères requis.

La Figure 29 décrit les différents composants d'une matrice d'usage.

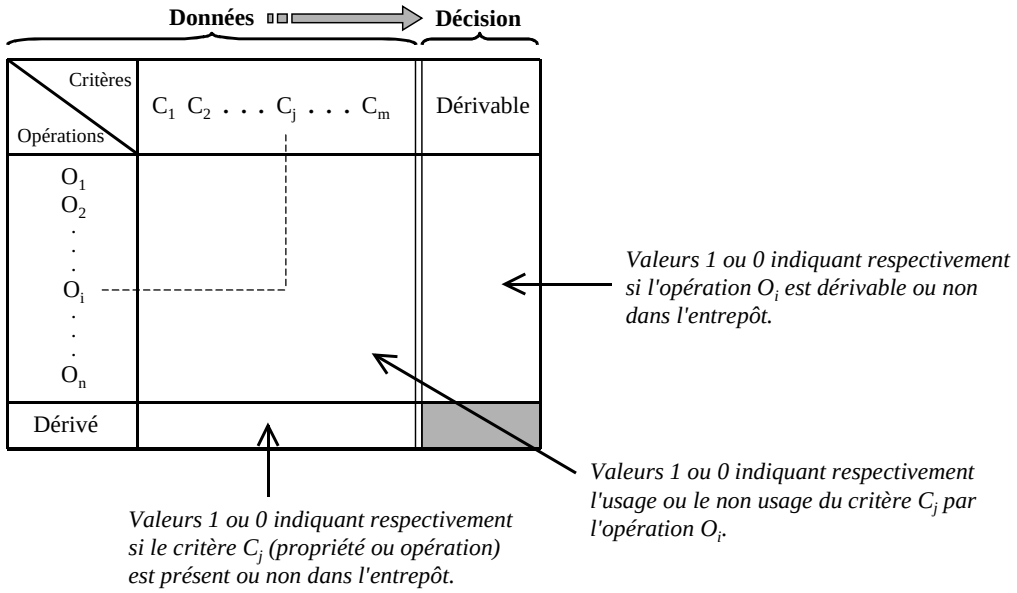


Figure 29 : Concept de matrice d'usage dans le contexte des entrepôts objet.

Nous utilisons deux matrices d'usage :

- la **matrice d'usage des propriétés** (MUP) se propose de déterminer si les propriétés nécessaires aux méthodes sont dérivées dans l'entrepôt,
- la **matrice d'usage des opérations** (MUO) sert à déterminer si les opérations utilisées par chaque opération sont disponibles dans l'entrepôt.

Par conséquent, le processus de définition des comportements s'appuie localement sur une MUP et globalement sur une MUO. Les deux sections suivantes présentent la construction et l'analyse des MUP et de la MUO.

3.3 Matrices des propriétés (MUP)

Cette section décrit la construction des Matrices d'Usage des Propriétés (MUP) ainsi que leur analyse. La définition des MUP s'effectue en deux étapes :

- la première étape consiste à construire la partie des données,
- la seconde étape analyse les données des MUP pour compléter la colonne supplémentaire utilisée lors de la décision pour la "*dérivabilité*" ou la non "*dérivabilité*".

3.3.1 Construction des MUP

Une MUP est élaborée pour chaque fonction de construction participant à l'élaboration d'un entrepôt. La construction des MUP est réalisée automatiquement lors de la définition des données et des structures des classes entrepôt (*Mapping*^{c}).

- Les lignes de la MUP correspondent à l'ensemble des opérations définissant le comportement des classes source impliquées dans la fonction de construction.
- Les colonnes de la MUP correspondent à l'ensemble des propriétés définissant la structure des classes source impliquées dans la fonction de construction.

- Chaque case (i,j) de la MUP contient la valeur 1 si la $i^{ième}$ opération utilise la $j^{ième}$ propriété ; elle contient la valeur 0 sinon.
- La ligne supplémentaire "*Dérivé*" indique si la propriété est dérivée dans l'entrepôt.

Remarque :
Une opération de jointure utilisée dans l'expression de la fonction de construction, nécessite un ajustement de la matrice. En effet, la jointure de deux classes source cs_1 et cs_2 engendre une classe entrepôt résultante *c* dont la structure est composée de l'union des structures de cs_1 et cs_2. Si que cs_1 et cs_2 sont reliées par une relation *r* (d'association ou de composition), la colonne correspondant à la relation *r* est supprimée de la matrice.

Remarque :
Il est également possible d'optimiser les MUP. Cette optimisation consiste à supprimer les colonnes comprenant uniquement des valeurs 0 (la propriété ne participe pas à la construction de la structure de la classe entrepôt et elle est utilisée par aucune opération source).

3.3.2 Analyse des MUP

Le processus automatique d'analyse des MUP s'effectue à l'aide d'un algorithme qui permet de préciser la "*dérivabilité*" des opérations d'une classe. Plus précisément, pour chaque opération, l'algorithme détermine si l'ensemble des propriétés est disponible (dérivé) dans l'entrepôt ; dans ce cas, la case de la colonne "*Dérivable*" prend la valeur 1, sinon elle prend la valeur 0. Tandis que la MUP est complétée, l'ensemble des éléments manquants est retourné en sortie de l'algorithme (cet ensemble est vide si l'opération est dérivable).

```
Algorithme. Analyse_Locale(i,MUP)
Entrées :
   i  : indice identifiant l'opération analysée dans la matrice
   MUP : matrice d'usage locale des propriétés
Sortie :
   ensemble des propriétés manquantes utilisées par i
Début
ens←∅;
pour j←1 à colonnes(MUP) faire
   si MUP[i,j]=1 alors
      si MUP[lignes(MU)+1,j]=0 alors ens←ens∪{j};
si (ens=∅) alors MUP[i,colonnes(MUP)+1]←1;
sinon MUP[i,colonnes(MUP)+1]←0;
retourne ens;
Fin.
```

Les fonctions *lignes(MUP)* et *colonnes(MUP)* retournent respectivement le nombre de lignes (ou opérations) et le nombre de colonnes (ou propriétés) composant la MUP.

3.3.3 Description des MUP de l'exemple complet

EXEMPLE : Le processus de définition du comportement est réalisé après celui des structures de données ; nous poursuivons l'exemple décrit dans la section 2.8. Caque fonction de construction donne lieu à la l'élaboration d'une MUP :

-Extraction des établissements → MUP1

-Extraction des services qui composent les établissements → MUP2

-Extraction des spécialistes en conservant la hiérarchie d'héritage source → MUP3

-Extraction des cabinets médicaux des spécialistes → MUP4

-Extraction des visites effectuées par les patients chez des spécialistes ainsi que des prescriptions ordonnées à cette occasion par les professionnels de santé → MUP5

La MUP1 permet de déterminer l'ensemble des opérations disposant des propriétés nécessaires dans l'entrepôt ; il s'agit des opérations *nbServices*, *ETABLISSEMENTbudget*, *ETABLISSEMENTestPublic, taxe, depart, region*.

La MUP2 permet de déterminer les opérations disposant des propriétés nécessaires dans l'entrepôt ; il s'agit des opérations *tailleEquipe*, *SERVICEnbServices*, *SERVICEbudget*.

La MUP3 permet de déterminer l'ensemble des opérations ayant chacune l'ensemble des propriétés (qu'elle manipule) dérivées dans l'entrepôt ; il s'agit des opérations *est_adulte*, *est_conventionne*, *taux_teletransmission*, *la_specialite*, *ratio_teletransmission* et *ratio_generique*.

L'analyse de la MUP4 permet d'obtenir les opérations *est_urbain*, *nb_praticiens*, *specialites* et *est_adulte*.

La MUP5 permet d'obtenir les opérations *est_adulte*, *VISITEtarif*, *montant_prescription*, *depassement*, *est_teletransmise*, *MEDICAMENTtarif* et *est_generique*.

	nom	adresse. libelle	adresse. code	adresse. ville	statut	subvention. montant	subvention. institution	subvention. type	organisation	**Dérivable**
nbServices()	0	0	0	0	0	0	0	0	1	1
budget()	0	0	0	0	0	0	0	0	1	1
estPublic()	0	0	0	0	1	0	0	0	0	1
taxe()	0	0	0	1	1	1	0	1	0	1
depart()	0	0	1	0	0	0	0	0	0	1
region()	0	0	1	0	0	0	0	0	0	1
Dérivé	1	1	1	1	1	1	1	1	1	

Tableau 12 : Représentation de la MUP1.

	nom	budget	etablissmt	equipe	directeur	**Dérivable**
tailleEquipe()	0	0	0	1	0	1
budget()	0	1	0	0	0	1
estPublic()	0	0	1	0	1	1
Dérivé	1	1	1	1	1	

Tableau 13 : Représentation de la MUP2.

colonnes pouvant être supprimées (optimisation)

	nom	prenom	adresse.libelle	adresse.code	adresse.ville	naissance.annee	naissance.mois	naissance.jour	num_prat	categorie	specialite	type_convention	diplomes	travaille	intervient	dirige	consultations	**Dérivable**
age()	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
est_adulte()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
est_conventionne()	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
nb_services_diriges()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
taux_teletransmission()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
la_specialite()	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
ratio_teletransmission()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
ratio_generique()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Dérivé	1	1	0	0	1	0	0	0	0	0	1	1	0	1	0	0	1	

Tableau 14 : Représentation de la MUP3.

	CABINETnom	CABINETadresse.code	CABINETadresse.ville	praticiens	naissance.annee	naissance.mois	naissance.jour	specialite	type_convention	dirige	consultations	**Dérivable**
est_rural()	0	1	0	0	0	0	0	0	0	0	0	0
est_urbain()	0	0	0	0	0	0	0	0	0	0	0	1
nb_praticiens()	0	0	0	1	0	0	0	0	0	0	0	1
specialites()	0	0	0	1	0	0	0	0	0	0	0	1
age()	0	0	0	0	1	1	1	0	0	0	0	0
est_adulte()	0	0	0	0	0	0	0	0	0	0	0	1
est_conventionne()	0	0	0	0	0	0	0	0	1	0	0	0
nb_services_diriges()	0	0	0	0	0	0	0	0	0	1	0	0
taux_teletransmission()	0	0	0	0	0	0	0	0	0	0	1	0
la_specialite()	0	0	0	0	0	0	0	1	0	0	0	0
ratio_teletransmission()	0	0	0	0	0	0	0	1	0	0	1	0
ratio_generique()	0	0	0	0	0	0	0	0	1	0	1	0
Dérivé	1	0	1	1	0	0	0	0	0	0	0	

Tableau 15 : Représentation optimisée de la MUP4.

	naissance.annee	naissance.mois	naissance.jour	specialite	type_convention	dirige	consultations	reference	honoraire	teletransmise	poids	taille	tension.max	tension.min	praticien	prescription	code	MEDICAMENTcategorie	tarif	taux_secu	type_molecule	generique	**Dérivable**
age()	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
est_adulte()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
est_conventionne()	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nb_services_diriges()	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
taux_teletransmission()	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
la_specialite()	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ratio_teletransmission()	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ratio_generique()	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
est_obese()	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
hypertension()	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
VISITEtarif()	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1
montant_prescription()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1
depassement()	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1
est_teletransmise()	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
MEDICAMENTtarif()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
est_generique()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Dérivé	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	1	1	1	1	1	

Tableau 16 : Représentation optimisée de la MUP5.

colonne supprimée à cause de la jointure entre PRATICIEN et VISITE.

3.4 Matrice des opérations (MUO)

Cette partie décrit la phase de construction de la MUO ainsi que son analyse pour compléter la colonne supplémentaire qui permet de décider de la dérivation ou non d'une opération.

3.4.1 Construction de la MUO

Une seule matrice d'usage des opérations MUO est construite pour tout l'entrepôt. Cette matrice est construite de la manière suivante :

- Chaque ligne de la MUO correspond à une opération définissant le comportement des classes source impliquées dans la fonction de construction ; seules les opérations "*dérivables*", en fonction du processus précédent, sont retenues.
- Chaque colonne de la MUO correspond à une opération définissant le comportement des classes source qui sont impliquées parmi toutes les fonctions de construction de l'entrepôt.
- Chaque case (i,j) de la MUO contient la valeur 1 si la $i^{ième}$ opération utilise la $j^{ième}$ opération sinon elle contient la valeur 0.

La ligne supplémentaire est fixée à –1 tant que l'opération n'a pas été analysée ; en effet il n'est pas possible à ce stade de savoir si l'opération est dérivée.

3.4.2 Analyse de la MUO

Le processus d'analyse par la MUO est plus complexe que l'analyse des MUP. Ceci provient du fait que l'on ne peut identifier au préalable les opérations manipulées et présentes dans l'entrepôt. Une autre difficulté est liée au cas particulier des appels mutuels entre opérations : ceux-ci génèrent alors un cycle dans le processus d'analyse ; l'intervention de l'administrateur est alors nécessaire pour indiquer si les opérations sont dérivables.

Le processus semi-automatique d'analyse de la MUO s'effectue à l'aide d'un algorithme qui permet de préciser la "*dérivabilité*" des opérations. Plus précisément, pour chaque opération i, l'algorithme détermine si l'ensemble des opérations utilisées par cette opération sont présentes dans l'entrepôt. Si l'opération j nécessaire à une opération i n'est pas encore analysée, le processus lance son analyse. Après l'analyse complète, si l'ensemble des opérations nécessaires sont dérivées, la case de la colonne "*Dérivable*" prend la valeur 1, sinon elle prend la valeur 0 et l'ensemble des opérations manquantes est retourné en sortie.

```
Algorithme. Analyse_Globale(i,MUO,visite)
Entrées :
   i      : indice identifiant l'opération analysée dans la matrice
   visite : vecteur détectant les cycles
Sortie :
   ensemble des opérations manquantes utilisées par l'opération i
Début
ens←∅;
pour j←1 à colonnes(MUO) faire
   début
   visite[i]←1;
   si MUO[i,j]=1 alors
      cas MUO[lignes(MUO)+1,j]=0 :  début // opération manquante
         ens←ens∪{j};
         fin;
      cas MUO[lignes(MUO)+1,j]=-1 : début // opération non analysée
         si visite[j]=1 alors ens←ens∪{j}; // détection d'un cycle
         sinon début // analyse
               visite[i]←1;
               MUP←determineMUP(j,MUO);
               jj←numero(j,MUO,MUP);
               ens=ens∪Analyse_Locale(jj,MUP);
               ens=ens∪Analyse_Globale(j,MUO,visite);
               fin;
         fin;
   fin;
si (ens=∅) alors MUO[i,colonnes(MUO)+1]←1;
sinon MUO[i,colonnes(MUO)+1]←0;
retourne ens;
Fin.
```

3.4.3 Description de la MUO de l'exemple complet

EXEMPLE : Nous poursuivons l'exemple précédent. De manière analogue aux MUP, la MUO est analysée en utilisant l'algorithme précédent, appliqué à chaque ligne.

	age()	est_adulte()	est_conventionne()	nb_services_diriges()	taux_teletransmission()	la_specialite()	ratio_teletransmission()	ratio_generique()	est_rural()	est_urbain()	nb_praticiens()	specialites()	prochaine_vaccination()	vaccination_a_jour()	est_obese()	hypertension()	VISITEtarif()	montant_prescription()	depassement()	est_teletransmise()	MEDICAMENTtarif()	est_generique()	nbServices()	taxe()	ETABLISSEMENTbudget()	ETABLISSEMENTestPublic()	tailleEquipe()	SERVICEbudget()	SERVICEestPublic()	**Dérivable**
MUP3est_adulte()	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MUP4est_adulte()	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MUP5est_adulte()	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
est_conventionne()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
nb_services_diriges()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
taux_teletransmission()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
la_specialite()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
ratio_teletransmission()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
ratio_generique()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
est_urbain()	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nb_praticiens()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
specialites()	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
VISITEtarif()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
montant_prescription()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
depassement()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
est_teletransmise()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
MEDICAMENTtarif()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
est_generique()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
nbServices()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
taxe()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
ETABLISSEMENTbudget()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
ETABLISSEMENTestPublic()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
tailleEquipe()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
SERVICEbudget()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
SERVICEestPublic()	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
Dérivé	0	-1	-1	0	-1	1	0	-1	0	-1	-1	-1	-1	-1	0	0	-1	-1	-1	1	0	1	-1	-1	-1	1	-1	-1	-1	

Tableau 17 : Représentation de la MUO.

3.5 Définition du comportement

En analysant les matrices d'usage locales (MUP) et la matrice d'usage globale (MUO), nous déterminons l'ensemble des opérations dérivables.

Lorsqu'une opération n'est pas dérivée, le processus indique les éléments (propriétés, opérations) nécessaires manquants, afin que l'administrateur puisse ajuster l'entrepôt à ses besoins en dérivant les éléments nécessaires à une opération qu'il souhaite extraire.

Le processus de définition du comportement des classes entrepôt est défini par l'algorithme suivant. Les analyses des matrices locales sont réalisées puis l'analyse de la matrice globale est effectuée. A chaque opération analysée localement ou globalement, l'ensemble des

éléments manipulés manquants est affiché à l'administrateur. Lorsque les analyses sont terminées, il est possible d'indiquer le comportement dérivé des classes entrepôt.

```
Algorithme. Deriver_Comportement()
Sortie :
   ensemble des méthodes dérivables
Début
// analyses locales (MUP, MUO)
pour chaque MUP faire
   pour i←1 à colonnes(MUP) faire
     si     ((manque←Analyse_Locale(i,MUP))≠∅)     alors     affiche(i,manque);
// préparation de l'analyse globale (MUO)
pour i←1 à colonnes(MUO) faire
   début
   MUP←determineMUP(i,MUO);
   ii←numero(i,MUO,MUP);
   si MUP[ii,colonnes(MUP)+1]=1 alors MUO[i,colonnes(MUO)+1]=0
   sinon MUO[i,colonnes(MUO)+1]=-1;
   fin;
// analyse globale (MUO)
pour i←1 à colonnes(MUO) faire
   début
   pour ii←1 à colonnes(MUO) faire visite[ii]←0;
     si ((manque←Analyse_Globale(i,MUO,visite))≠∅)
     alors affiche(i,manque);
   fin;
// décision comportement dérivable
res←∅;
pour i←1 à colonnes(MUO) faire
   début
   MUP←determineMUP(i,MUO);
   ii←numero(i,MUO,MUP);
   si MUP[ii,colonnes(MUP)+1]=1 &&
      MUO[i,colonnes(MUO)+1]=1 alors res←res∪{i};
      fin;
retourner res;
Fin.
```

Remarque :
Une fonction de construction est définie au travers d'une expression constituée d'opérations pouvant donner lieu à plusieurs classes entrepôt (il s'agit de l'extension H ou C visant à conserver les hiérarchies existantes). Dans ce cas particulier, la détermination des opérations dérivables s'opère sur la classe intermédiaire qui est ensuite divisée en plusieurs classes entrepôt conformément à hiérarchie reconstruite.

EXEMPLE : Nous reprenons l'exemple des sections précédentes. En s'appuyant sur les matrices d'usage locales MUP1, MUO1, MUP2, MUP3, MUP4, MUP5 et la matrice d'usage globale MUO, il est alors possible de déterminer le comportement dérivé des classes entrepôt produites. Les

classes entrepôt correspondent aux définitions de l'exemple complet présenté au chapitre II dans la section 4.5.

3.6 Synthèse

Cette section a étudié le principe d'extraction du comportement des classes que l'administrateur spécifie dans l'entrepôt. Notre approche s'inspire des matrices d'usage proposées dans les bases de données objet réparties en adaptant cette technique au contexte des entrepôts de données orientées objet.

- Nous avons défini des matrices d'usage locales déterminant les opérations dérivables en fonction des propriétés (MUP) dérivées.
- Nous avons défini une matrice d'usage globale déterminant les opérations dérivables en intégrant l'invocation des autres méthodes dérivées (MUO).

4 CONCLUSION

Dans ce chapitre, nous définissons le processus de construction de l'entrepôt de données à partir d'une source globale. L'intérêt de notre approche réside dans le fait qu'elle aborde tant les aspects statiques (au travers de l'extraction des données et des structures de données) que les aspects dynamiques (au travers de l'extraction des comportements des données).

- Nous proposons de définir le **processus de définition des structures** des classes entrepôt, par l'intermédiaire d'une fonction de construction (*Mapping*^{c}). Cette fonction est exprimée à partir d'opérations de base, comprenant
  - des opérations de **qualification** (sélection, jointure, groupement, dégroupement) définissant des critères qualifiant les objets source pertinents à dériver dans l'entrepôt pour calculer l'extension des classes entrepôt,
  - des opérations de **structuration** (projection, masquage, accroissement) définissant les propriétés dérivées et calculées à partir de la source ou les propriétés spécifiques constituant la structure des classes entrepôt,
  - des opérations **ensemblistes** (union, intersection, différence) combinant plusieurs classes source afin de définir une nouvelle classe entrepôt,
  - des opérations de **hiérarchisation** (généralisation, spécialisation) réorganisant la hiérarchie d'héritage des classes entrepôt en fonction des besoins spécifiques de l'entrepôt.
- Nous proposons de définir le **processus de définition semi-automatique des comportements** des classes entrepôt. Pour dériver les opérations d'une classe source, nous déterminons si l'ensemble des propriétés et des opérations requises par chaque opération est présent dans l'entrepôt. Le processus de définition des comportements s'appuie sur :
  - une **matrice locale** pour chaque fonction de construction afin d'identifier les propriétés (MUP) utilisées par les opérations et

- une **matrice globale** pour tout l'entrepôt afin d'identifier les appels entre les différentes opérations (MUO).

Le chapitre II et ce chapitre III définissent les concepts permettant de modéliser les données de l'entrepôt, pertinentes pour les décideurs. Néanmoins, il reste à définir un ensemble d'opérateurs permettant de manipuler l'information contenue dans l'entrepôt. Ce langage de manipulation doit prendre en compte les concepts spécifiques de notre modélisation des entrepôts.

CHAPITRE IV :
UN LANGAGE DE MANIPULATION DES DONNEES DE L'ENTREPOT

1 INTRODUCTION A LA MANIPULATION DES OBJETS DANS L'ENTREPOT

1.1 Objectif

Nous avons défini un modèle conceptuel spécifique pour les entrepôts de données complexes, temporelles et archivées. Ce chapitre propose un langage de manipulation pour exploiter les concepts spécifiques de notre modélisation.

Ce langage est destiné aux utilisateurs experts qui effectuent des interrogations directement sur l'entrepôt de données. En effet, il est parfois utile d'obtenir rapidement des informations ou bien d'effectuer des analyses ponctuelles directement sur l'ensemble des données décisionnelles stockées dans l'entrepôt.

Pour cela, nous définissons une algèbre (c'est à dire un ensemble d'opérateurs servant de base à l'implantation de langages textuels ou graphiques) qui prend en compte les spécificités inhérentes à notre modèle de données pour les entrepôts. En particulier, il est nécessaire de pouvoir manipuler les données :

- organisées sous forme d'objets entrepôt constitués de plusieurs états (courants, passés et archivés),
- organisées sous forme d'états indépendamment les uns des autres,
- organisées sous la forme d'un ensemble d'états chronologiquement ordonnés.

1.2 Existant

Dans le domaine des entrepôts de données, les travaux de recherche concernant les langages d'interrogation se focalisent essentiellement sur la manipulation de concepts issus des modèles multidimensionnels [Agrawal, et al. 1997] [Gyssen, et al. 1997] [Lehner, et al. 1998] [Pedersen, Jensen 1999]. Or, dans notre architecture, la modélisation de l'entrepôt n'intègre pas ces aspects multidimensionnels. Cette organisation, orientée décision, est utilisée au niveau des magasins (ceci fait l'objet du chapitre V). D'autres approches proposent des extensions du langage standard SQL ; nous pouvons citer principalement le langage *RISQL* [RedBrick 1998]. Cependant, ces travaux ne traitent pas de l'aspect temporel des données dans les entrepôts (aspect pourtant essentiel dans le contexte de l'aide à la décision).

Dans le domaine des bases de données temporelles, de très nombreux travaux [Tsotras, Kumar 1996] proposent d'étendre les langages de manipulation aux aspects temporels. TSQL2 [Snodgrass 1995] constitue une étude importante dont l'essentiel des propositions est intégré au standard SQL3 [Snodgrass, et al 1998]. Plusieurs projets visent à étendre le standard des bases objet proposé par l'ODMG : T_ODMG [Bertino, et al 1998a], TEMPOS [Canavaggio 1997], TOOBIS [Toobis 1998a, 1998b, 1998c], TAU [Kakoudakis, Theodoulidis 1996]. Cependant, ces travaux ne prennent pas en compte les concepts inhérents à notre modèle. En particulier, notre modélisation de données intègre un mécanisme d'archivage nécessitant des opérateurs de manipulation adaptés.

1.3 Proposition

Ce chapitre présente l'algèbre associée au modèle de données décrit dans le chapitre II. Cette algèbre s'inspire des principales algèbres objet et notamment de celle définie par Shaw et Zdonik dans [Shaw, Zdonik 1990]. L'algèbre que nous définissons doit permettre d'interroger directement toute l'information contenue dans l'entrepôt. Par conséquent, en plus des spécificités inhérentes à l'approche orientée objet, notre algèbre doit prendre en compte les caractéristiques des objets entrepôt et aux différents états des objets entrepôt.

Cette algèbre se compose d'un ensemble d'opérateurs d'interrogation. Nous reprenons les opérations algébriques des algèbres pour objets temporels que nous adaptons aux objets entrepôt ; ces opérateurs sont étendus afin d'intégrer le concept d'état. Nous définissons de nouveaux opérateurs, spécifiques à notre modélisation, pour manipuler les différentes catégories d'états qui composent les objets entrepôt. Ils permettent également de transformer les données en séries temporelles d'états afin d'offrir différentes perspectives sur les informations et leurs évolutions de manière à faciliter la compréhension et l'analyse du contenu de l'entrepôt.

Nos propositions ne préjugent en rien des aspects d'optimisation de requête que nous n'étudions pas dans cette thèse.

Ce chapitre est divisé en quatre sections.

- La section 2 reprend les principaux opérateurs des algèbres objet pour les adapter à notre modélisation de l'information dans l'entrepôt.
- La section 3 décrit de nouveaux opérateurs permettant d'interroger les différents états des objets entrepôt..
- La section 4 présente l'intégration de l'aspect temporel des données en offrant des opérations pour la manipulation du domaine temporel des états.
- La section 5 introduit une opération de formatage des états en série temporelle, autorisant des traitements et des transformations facilitant la compréhension des évolutions de valeur des objets.

2 ADAPTATION DES OPERATEURS OBJET A NOTRE MODELE

Nous rappelons qu'une classe entrepôt c est définie par (Nom^c, $Type^c$, $Super^c$, $Extension^c$, $Mapping^c$, $Tempo^c$, $Archi^c$) où $Extension^c=\{o^1, o^2,\ldots, o^x\}$ est l'extension de la classe. Un objet entrepôt o_i est défini par (oid^i, S^i_0, $Histoire^i$, $Archive^i$, $Origine^i$) où

- oid^i est son identifiant,
- $S^i_0=(DomT^i_0, V^i_0)$ est son état courant,
- $Histoire^i=\{S^i_{p1}, S^i_{p2},\ldots, S^i_{ppi}\}$ est l'ensemble de ses états passés,
- $Archive^i=\{S^i_{a1}, S^i_{a2},\ldots, S^i_{aai}\}$ est l'ensemble de ses états archivés et
- $Origine^i$ est l'ensemble des objets source dont il est issu.

Chaque état S^i_j représente la valeur ou une partie de la valeur V^i_j de l'objet o^i aux instants appartenant au domaine temporel associé $DomT^i_j$ (le chapitre II présente en détail le modèle).

Nous redéfinissons dans cette section les principaux opérateurs des algèbres pour objet afin de tenir compte de notre modélisation.

2.1 Opérateurs ensemblistes

Nous adoptons les opérateurs ensemblistes de l'algèbre objet **Encore** [Shaw, Zdonik 1990]. En effet, les requêtes de manipulation sont exprimées au travers d'opérateurs d'interrogation qui requièrent une puissance d'expressivité suffisante pour manipuler et transformer des ensembles d'éléments.

Tous ces opérateurs de manipulation des ensembles peuvent être, soit des ensembles d'objets entrepôt, soit des ensembles d'états. Si l'opérateur est appliqué à un ensemble d'objets entrepôt, alors le résultat est formé d'objets entrepôt ; si l'opérateur est appliqué à un ensemble d'états alors le résultat est formé d'états.

Les opérateurs ensemblistes réalisent l'union, l'intersection ou la différence entre deux ensembles. Ces opérateurs étant basés sur l'égalité, deux types d'union, d'intersection et de différence sont définis : un basé sur l'égalité d'identifiant (notée $=_I$) pour les objets entrepôt, l'autre basé sur l'égalité de valeur (notée =) pour les objets entrepôt ou bien pour les états.

<u>*Définition*</u> :
Les opérateurs ensemblistes permettent de combiner des ensembles d'éléments (pouvant être soit des objets entrepôt, soit des états) en réalisant l'union, l'intersection ou la différence. La syntaxe des opérateurs ensemblistes est la suivante :

$$op_{ens}(Ens^1, Ens^2)=Ens$$

avec $op_{ens} \in$ {V_Union, V_Intersect, V_Difference, I_Union, I_Intersect, I_Difference}

<u>Entrées</u>
Ens^1 et Ens^2 sont deux ensembles d'éléments (pouvant être soit des états soit des objets entrepôt).

<u>Sortie</u>
Ens est l'ensemble des éléments (états ou objets entrepôt) tel que

- Si op_{ens}=*V_Union*, alors $Ens=\{e \mid \exists e \in Ens^1 \vee \exists e \in Ens^2 \wedge (\neg (\exists e' \in Ens \mid e=e'))\}$,
- Si op_{ens}=*I_Union*, alors $Ens=\{e \mid \exists e \in Ens^1 \vee \exists e \in Ens^2 \wedge (\neg (\exists e' \in Ens \mid e=_I e'))\}$,
- Si op_{ens}=*V_Intersect*, alors $Ens=\{e \mid \exists\, e \in Ens^1 \wedge (\exists e' \in Ens^2 \mid e=e')\}$,
- Si op_{ens}=*I_Intersect*, alors $Ens=\{e \mid \exists\, e \in Ens^1 \wedge (\exists e' \in Ens^2 \mid e=_I e')\}$,
- Si op_{ens}=*V_Difference*, alors; $Ens=\{e \mid \exists e \in Ens^1 \wedge (\neg(\exists e' \in Ens^2 \mid e=e'))\}$,
- Si op_{ens}=*I_Difference*, alors; $Ens=\{e \mid \exists e \in Ens^1 \wedge (\neg(\exists e' \in Ens^2 \mid e=_I e'))\}$.

De manière informelle,

- l'union consiste à calculer l'ensemble *Ens* des éléments (états ou objets entrepôt) appartenant à au moins un des deux ensembles Ens^1 et Ens^2,

- l'intersection consiste à calculer l'ensemble *Ens* des éléments (états ou objets entrepôt) appartenant aux deux ensembles Ens^1 et Ens^2,
- la différence consiste à calculer l'ensemble *Ens* des éléments (états ou objets entrepôt) appartenant à l'ensemble Ens^1 et n'appartenant pas à l'ensemble Ens^2

Remarque :
Il est important de remarquer que l'égalité d'identifiant n'est pas disponible pour les ensembles d'états.

2.2 Traitements sur les ensembles

Nous adoptons l'opérateur d'aplatissement des ensembles d'ensembles. L'intérêt de cet opérateur est qu'il supprime un niveau d'imbrication des ensembles.

Définition :
L'opérateur d'aplatissement *Flatten* permet d'aplatir un ensemble d'ensemble (d'éléments pouvant être des états ou des objets entrepôt). La syntaxe de l'opérateur *Flatten* est la suivante :

Flatten(Ens)=Ens'

Entrée
$Ens=\{Ens^1,\ldots, Ens^x\}$ est un ensemble d'ensembles tels que $\forall i\in[1..x]$, $Ens^i=\{e^i_1,\ldots, e^i_{ni}\}$.

Sortie
$Ens=\{e^1_1,\ldots, e^1_{n1},\ldots, e^x_1,\ldots, e^x_{nx}\}$ est l'ensemble des éléments de $Ens^1,\ldots, Ens^x$.

Enfin, nous adoptons un opérateur permettant d'éliminer les doubles dans un ensemble et un opérateur d'élimination des ensembles vides dans un ensemble d'ensembles.

Définition :
L'opérateur *DupEliminate* permet d'éliminer les doublons d'un ensemble d'éléments (états ou objets entrepôt) suivant l'égalité de valeur. La syntaxe de l'opérateur *DupEliminate* est la suivante :

DupEliminate(Ens)=Ens'

Entrée
Ens est un ensemble d'éléments,

Sortie
$Ens'=\{e_1, \ldots, e_x\}$ est un ensemble sans doublon tel que $\forall i\in[1..x]$, $\forall j\neq i\in[1..x]$, $\neg e_i=e_j$.

Définition :
L'opérateur *EmptyEliminate* permet d'éliminer les ensembles vides dans un ensemble d'ensembles. La syntaxe de l'opérateur *EmptyEliminate* est la suivante :

EmptyEliminate(Ens)=Ens'

Entrée
Ens est un ensemble d'ensembles.

Sortie
$Ens'=\{Ens^1, Ens^2,\ldots, Ens^x\}$ est un ensemble d'ensembles tel que $\forall i\in[1..x]$, $Ens^i\neq\emptyset$.

2.3 Opérateurs d'interrogation

Notre algèbre reprend les opérateurs classiques en base de données [Chrisment, et al 1999] de sélection, de projection, de jointure, de groupement et dégroupement. Dans note contexte, les opérateurs sont étendus pour s'appliquer indifféremment aux objets entrepôt ou aux états qui les composent.

2.3.1 Opérateur Project

La projection permet de réduire le nombre des propriétés et d'opérations d'un ensemble d'éléments de même type, pouvant être soit des objets entrepôt, soit des états.

Définition :
L'opérateur *Project* permet de projeter les propriétés et opérations désirées pour un ensemble d'éléments (états ou objets entrepôt). La syntaxe de l'opérateur *Project* est la suivante :

Project(x, Ens, x.chem$_1$,…, x.chem$_n$)=Ens'

Entrée
x variable d'itération sur les éléments de l'ensemble désigné *Ens*,

Ens est un ensemble d'éléments pouvant être, soit des états, soit des objets entrepôt,

$\forall i\in[1..n]$, $chem_i$ est un chemin désignant une propriété ou une opération projetée.

Sortie
Ens' est l'ensemble de n-uplets formés des propriétés ou opérations désignées par les différentes expressions de chemin tel que

- Si $Ens=\{S_1, S_2,\ldots, S_n\}$ est un ensemble d'états, alors $Ens'=\{[p_1{:}S_i.chem_1, p_2{:}S_i.chem_2,\ldots, p_n{:}S_i.chem_n] \mid \exists S_i\in Ens\}$,
- Si $Ens=\{o^1, o^2,\ldots, o^x\}$ est un ensemble d'objets entrepôt, alors $Ens'=\{[p_1{:}o^i.chem_1, p_2{:}o^i.chem_2,\ldots, p_n{:}o^i.chem_n] \mid \exists o^i\in Ens\}$.

Remarque :
Dans le contexte de notre modèle d'entrepôts, l'expression d'un chemin $x.chem_i$ permet, à partir d'un élément, d'accéder à d'autres éléments via une succession d'opérations traduisant une relation de composition ou d'association.

- Lorsque le chemin est de la forme *x* (le nombre d'opérations du chemin est nul), le chemin correspond à l'élément lui-même.

- Lorsque le chemin est de la forme *x.op*, le chemin *op* correspond à un attribut ou une opération défini(e) sur le type de l'élément *x*.
- Lorsque le chemin est de la forme $x.op_1.\ldots.op_m$, ce chemin est tel que *x* correspond à l'élément de départ (il peut être un objet entrepôt, un état ou un n-uplet), op_1 correspond à une relation (d'association ou de composition) ou bien à une opération et $\forall i \in [2..m]$, op_i correspond à une opération définie pour les éléments de type de $i.op1.\ldots.op_{i-1}$.

EXEMPLE : Nous considérons les classes entrepôt *Etablissements* et *Services* (la classe *Etablissements* représente les hôpitaux et la classe *Services* représente les services qui composent les hôpitaux) de l'exemple complet d'entrepôt décrit au chapitre II dans la section 4.5. Nous utilisons ces deux classes comme illustration tout au long de ce chapitre.

L'extension de *Services* comprend les objets $\{o^1, o^2, o^3, o^4, o^5, o^6\}$ dont les valeurs sont décrites de la manière suivante :

o_1
nom:"chirurgie"
budget:18.000.000,00
etablismt:o_7
domT:<[07-2000, *now*]>

Histoire: { Budget:8.000.000,00 domT:<[01-2000,06-2000]> ; Budget:9.500.000,00 domT:<[07-1999,12-1999]> ; Budget:7.000.000,00 domT:<[01-1999,06-1999]> }

Archive: { Budget:6.500.000,00 domT:<[1997,1998]> ; Budget:6.000.000,00 domT:<[1995,1996]> }

o_2
nom:"ophtalmologie"
budget:2.000.000,00
etablismt:o_7
domT:<[07-2000, *now*]>

Histoire: { Budget:2.400.000,00 domT:<[01-2000,06-2000]> ; Budget:1.900.000,00 domT:<[07-1999,12-1999]> ; Budget:2.000.000,00 domT:<[01-1999,06-1999]> }

Archive: { Budget:2.500.000,00 domT:<[1997,1998]> ; Budget:2.000.000,00 domT:<[1995,1996]> }

o_3
nom:"urgences"
budget:15.000.000,00
etablismt:o_7
domT:<[07-2000, *now*]>

Histoire: { Budget:10.000.000,00 domT:<[01-1999;02-1999]> ; Budget:12.000.000,00 domT:<[03-1999;06-1999]; [10-1999;12-1999]> ; Budget:14.000.000,00 domT:<[07-1999;09-1999]; [01-2000;02-2000]> ; Budget:16.000.000,00 domT:<[03-2000;06-2000]> }

Archive: { Budget:17.500.000,00 domT:<[1997,1998]> ; Budget:15.000.000,00 domT:<[1995,1996]> }

o_4
nom:"chirurgie"
budget:12.000.000,00
etablismt:o_8
domT:<[03-2000, *now*]>

Histoire: { Budget:11.000.000,00 domT:<[06-1999,02-2000]> ; Budget:10.000.000,00 domT:<[01-1999,06-1999]> }

Archive: { Budget:8.500.000,00 domT:<[1997,1998]> ; Budget:8.000.000,00 domT:<[1995,1996]> }

o_5
nom:"urgences"
budget:25.000.000,00
etablismt:o_8
domT:<[07-2000, *now*]>

Histoire: { Budget:20.000.000,00 domT:<[01-2000,06-2000]; [01-1999,08-1999]> ; Budget:23.000.000,00 domT:<[07-1999,12-1999]> }

Archive: { Budget:20.000.000,00 domT:<[1997,1998]> ; Budget:18.000.000,00 domT:<[1995,1996]> }

o_6
nom:"chirurgie"
budget:30.000.000,00
etablismt:o_9
domT:<[01-2000, *now*]>

Histoire: { Budget:25.000.000,00 domT:<[01-1999,12-1999]> }

Archive: { Budget:23.500.000,00 domT:<[1997,1998]> ; Budget:20.000.000,00 domT:<[1995,1996]> }

L'extension de *Etalissements* comprend les objets {o^7, o^8, o^9 } dont les valeurs sont décrites de la manière suivante :

o_7
nom:"Hôpital Hôtel Dieu"
adresse:[libelle:"1, place Paris Notre Dame",
code:75004
ville:"Paris"]
statut:"public"
subventions:[montant:2.000.000,00,
institution:"ministère de la santé",
type:"public"]
organisation:{o_1, o_2, o_3}
domT:<[08-2000, *now*]>

Histoire: {
subventions:[montant:10.000.000,00, institution:"Conseil Régional Ile-de-France", type:"public"] domT:<[04-2000,07-2000]> ;
subventions:[montant:10.000.000,00, institution:"Groupe Fabre", type:"privée"] domT:<[02-2000,03-2000]> ;
subventions:[montant:1.000.000,00, institution:"Arc", type:"association"] domT:<[10-1999,01-2000]> ;
subventions:[montant:5.000.000,00, institution:"Conseil Régional Ile-de-France", type:"public"] domT:<[06-1999,09-1999]> ;
subventions:[montant:10.000.000,00, institution:"Groupe Fabre", type:"privée"] domT:<[03-1999,05-1999]> ;
subventions:[montant:1.000.000,00, institution:"Arc", type:"association"] domT:<[01-1999,02-1999]>
}

Archive: { montant:10.000.000,00, domT:<[1997,1998]> ; montant:8.000.000,00, domT:<[1995,1996]> }

o_8

nom:"Hôpital du Val de Grâce"
adresse:[libelle:"74, boulevard Port Royal",
code:75005
ville:"Paris"]
statut:"public"
subventions:[montant:20.000.000,00,
institution:"ministère de la santé",
type:"public"]
organisation:{o_4, o_5}
domT:<[01-2000, *now*]>

Histoire:
- subventions:[montant:14.000.000,00, institution:"Groupe Fabre", type:"privée"] domT:<[08-1999,12-1999]>
- subventions:[montant:3.000.000,00, institution:"Asso. maladies orphelines", type:"association"] domT:<[05-1999,07-1999]>
- subventions:[montant:18.000.000,00, institution:" Conseil Régional Ile-de-France ", type:"privée"] domT:<[01-1999,04-1999]>

Archive:
- montant:15.000.000,00, domT:<[1997,1998]>
- montant:10.000.000,00, domT:<[1995,1996]>

o_9

nom:"Clinique du Sport"
adresse:[libelle:"36, boulevard St Marcel",
code:75005
ville:"Paris"]
statut:"privée"
subventions:[montant:15.000.000,00,
institution:"Ministère de la Santé",
type:"public"]
organisation:{o_6}
domT:<[06-2000, *now*]>

Histoire:
- subventions:[montant:500.000,00, institution:"Groupe Fabre", type:"privée"] domT:<[04-2000,05-2000]>
- subventions:[montant:20.00.000,00, institution:"Conseil Régional Ile-de-France", type:"public"] domT:<[01-2000,03-2000]>
- subventions:[montant:3.500.000,00, institution:"Arc", type:"association"] domT:<[08-1999,12-1999]>
- subventions:[montant:20.00.000,00, institution:"Ministère de la Santé", type:"public"] domT:<[04-1999,07-1999]>
- subventions:[montant:18.00.000,00, institution:"Ministère de la Santé", type:"public"] domT:<[01-1999,03-1999]>

Archive:
- montant:20.000.000,00, domT:<[1997,1998]>
- montant:18.000.000,00, domT:<[1995,1996]>

Un administrateur peut exprimer la requête suivante pour "*obtenir le nom, la localisation, le statut des établissements hospitaliers*" :

Project(x, Etablissement, x.nom, x.adresse.ville, x.statut)

Le résultat obtenu est de type { [String, String, String] } et contient les valeurs suivantes :

{ ["Hôpital Hôtel Dieu", "Paris", "public"],
["Hôpital du Val de Grâce", "Paris", "public"],
["Clinique du Sport", "Paris", "privée"] }

2.3.2 Opérateur Select

La sélection appliquée sur un ensemble d'éléments (pouvant être des objets entrepôt ou des états) consiste à retenir un sous-ensemble des éléments vérifiant un prédicat.

Définition :
L'opérateur *Select* permet de sélectionner un ensemble d'éléments (soit des objets entrepôt, soit des états) satisfaisant à un prédicat. La syntaxe de l'opérateur *Select* est la suivante :

Select(x, Ens, P)=Ens'

Entrées
x variable d'itération sur les éléments de l'ensemble désigné *Ens*,

Ens est un ensemble d'éléments pouvant être soit des états soit des objets entrepôt,

P est un prédicat de sélection.

Sortie
Ens' est l'ensemble des éléments sélectionnés tel que

- Si $Ens=\{S_1, S_2,\ldots, S_n\}$ est un ensemble d'états, alors $Ens'=\{S_i \mid S_i \in Ens \wedge P(S_i)\}$ est l'ensemble des états satisfaisant au prédicat *P*,
- Si $Ens=\{o^1, o^2,\ldots, o^x\}$ est un ensemble d'objets entrepôt, alors $Ens'=\{o^i \mid o^i \in Ens \wedge P(o^i)\}$ est l'ensemble des objets entrepôt satisfaisant au prédicat *P*.

EXEMPLE : Nous poursuivons l'exemple précédent. Cette fois l'administrateur désire "*obtenir le nom, la localisation, le statut des établissements hospitaliers dont le montant de la dernière subvention est supérieur à 10.000.000,00 de francs*".

Project(x, **Select**(xx, Etablissement, xx.subventions.montant>10.000.000,00), x.nom, x.adresse.ville, x.statut)

Le résultat obtenu est de type { [String, String, String] } et contient les valeurs suivantes :

{ ["Hôpital du Val de Grâce", "Paris", "public"], ["Clinique du Sport", "Paris", "privée"] }

2.3.3 Opérateur Join

La jointure s'applique aussi sur un ensemble d'éléments (des objets entrepôt ou des états).

Définition :
La jointure entre deux ensembles est décrit de la manière suivante :

$\text{Join}(x_1, \text{Ens}^1, x_2, \text{Ens}^2, P)=\text{Ens}'$

Entrées
Ens^1 et Ens^2 sont deux ensembles d'éléments pouvant être soit des états soit des objets entrepôt,

x_1 et x_2 sont les variables d'itération sur les éléments des ensembles désignés Ens^1 et Ens^2,

P est un prédicat de jointure.

Sortie
Ens' est l'ensemble des éléments tel que

- Si $Ens^1=\{S^1_1, S^1_2,\ldots, S^1_n\}$ et $Ens^2=\{S^2_1, S^2_2,\ldots, S^2_n\}$ sont des ensembles d'états, alors $Ens'=\{[S^1_i, S^2_j] \mid S^1_i \in Ens^1 \wedge S^2_j \in Ens^2 \wedge P(S^1_i, S^2_j)\}$,
- Si $Ens^1=\{o^1_1, o^1_2,\ldots, o^1_n\}$ et $Ens^2=\{o^2_1, o^2_2,\ldots, o^2_n\}$ sont des ensembles d'objets entrepôt, alors $Ens'=\{[o^1_i, o^2_j] \mid o^1_i \in Ens^1 \wedge o^2_j \in Ens^2 \wedge P(S^1_i, S^2_j)\}$.

2.3.4 Opérateurs Nest et UnNest

Notre algèbre adapte les opérateurs de groupement et de dégroupement, connus respectivement sous le nom d'opérateur "*nest*" et "*unnest*" [Schek, Pistor 1982].

<u>*Définition*</u> :
L'opérateur *Nest* permet regrouper les éléments (pouvant être soit des objets entrepôt, soit des états) ayant la même valeur pour un attribut multivalué. La syntaxe de l'opérateur *Nest* est la suivante :

Nest(Ens, att)=Ens'

<u>Entrées</u>
Ens est un ensemble d'éléments (états ou objets entrepôt) dont le type de chacun contient l'attribut de groupement,

att désigne l'attribut sur lequel s'applique le groupement.

<u>Sortie</u>
Ens' est l'ensemble des n-uplets correspondant au critère de regroupement tel que

- Si $Ens=\{S_1, S_2,\ldots, S_n\}$ est un ensemble d'états, alors $Ens'=\{[p_1{:}S_i.p_1, p_2{:}S_i.p_2,\ldots, att{:}S_i.t,\ldots, p_n{:}S_i.p_n] \mid \forall r\in t, \exists S_i\in Ens, S_i.att=r\}$,
- Si $Ens=\{o^1, o^2,\ldots, o^x\}$ est un ensemble d'objets entrepôt, alors $Ens'=\{[p_1{:}o^i.p_1, p_2{:}o^i.p_2,\ldots, att{:}o^i.t,\ldots, p_n{:}o^i.p_n] \mid \forall r\in t, \exists o^i\in Ens, o^i.att=r\}$.

L'opérateur *UnNest* a l'effet opposé de celui de *Nest* (mais il ne s'agit pas de son inverse au sens mathématique).

<u>*Définition*</u> :
L'opérateur *UnNest* permet de dégrouper les éléments (pouvant être soit des objets entrepôt, soit des états) ayant une collection de valeurs communes. La syntaxe de l'opérateur *UnNest* est la suivante :

UnNest(Ens, att)=Ens'

<u>Entrées</u>
Ens est un ensemble d'éléments (états ou objets entrepôt) dont le type de chacun contient l'attribut de dégroupement,

att désigne l'attribut sur lequel s'applique le dégroupement.

<u>Sortie</u>
Ens' est l'ensemble des éléments formé des n-uplets correspondant au critère de dégroupement tel que

- Si $Ens=\{S_1, S_2,\ldots, S_n\}$ est un ensemble d'états, alors $Ens'=\{[p_1{:}S_i.p_1, p_2{:}S_i.p_2,\ldots, att{:}S_i.t,\ldots, p_n{:}S_i.p_n] \mid t\in S_i.att \wedge S_i\in Ens\}$,
- Si $Ens=\{o^1, o^2,\ldots, o^x\}$ est un ensemble d'objets entrepôt, alors $Ens'=\{[p_1{:}o^i.p_1, p_2{:}o^i.p_2,\ldots, att{:}o^i.t,\ldots, p_n{:}o^i.p_n] \mid t\in o^i.att \wedge o^i\in Ens\}$.

2.4 Synthèse

Nous avons adapté au concept spécifique d'objet entrepôt les principaux opérateurs habituellement rencontrés dans les algèbres à objet et temporelles. Nous avons adopté les opérateurs ensemblistes traditionnels (*V_Union*, *V_Intersect*, *V_Difference*, *I_Union*, *I_Intersect*, *I_Difference*, *Flatten*). Nous avons adopté les principaux opérateurs de manipulation des algèbres pour objet (*Project*, *Select*, *Join*, *Nest*, *UnNest*).

3 MANIPULATION DES ETATS ET DES OBJETS ENTREPOT

3.1 Opérateurs d'accès aux états Current, Past et Archive

Nous avons défini des opérateurs ensemblistes mais il convient de fournir d'autres opérateurs, spécifiques à notre modèle, pour manipuler les différents états des objets entrepôt. Chaque état des objets entrepôt est accessible au travers de trois opérateurs :

- un opérateur *Current* permettant d'obtenir l'ensemble des états courants,
- un opérateur *Past* permettant d'obtenir un ensemble d'ensembles des états passés,
- un opérateur *Archive* permettant d'obtenir un ensemble d'ensembles des états archivés.

La Figure 30 illustre le fonctionnement des opérateurs *Current*, *Past* et *Archive* où $Ens=\{o^1, o^2,\ldots,o^x\}$ est un ensemble d'objets entrepôt et $\forall i \in [1..x]$, S^i_0 est l'état courant de chaque objet entrepôt o^i, $\{S^i_{p1}, S^i_{p2},\ldots,S^i_{pi}\}$ est l'ensemble des états passés de chaque objet entrepôt o^i et $\{S^i_{a1}, S^i_{a2},\ldots,S^i_{ai}\}$ est l'ensemble des états archivés de chaque objet entrepôt o^i.

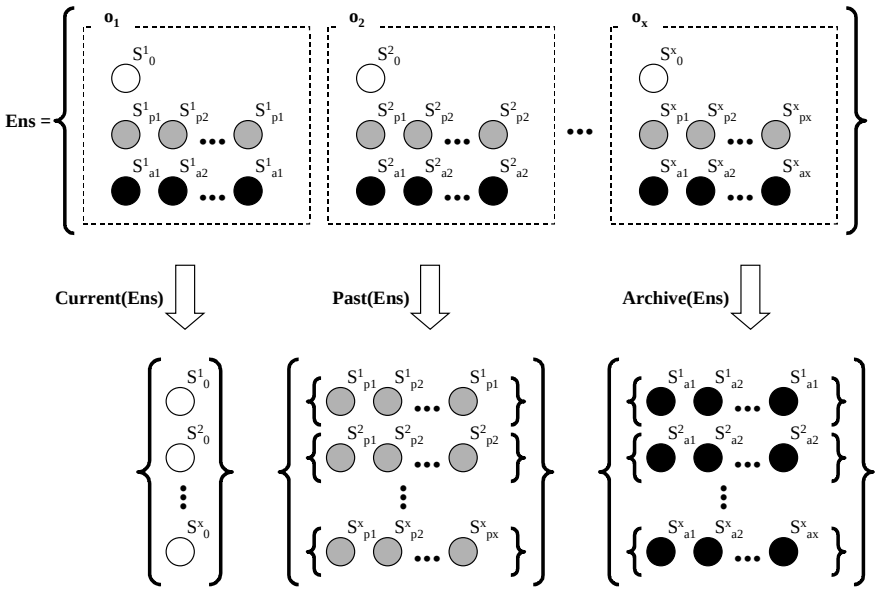


Figure 30 : Principes des opérateurs *Current*, *Past* et *Archive*.

3.1.1 Opérateurs d'accès aux états courants : Current

<u>*Définition*</u> :
L'opérateur *Current* permet d'accéder aux états courants d'un ensemble d'objets entrepôt. La syntaxe de l'opérateur *Current* est la suivante :

Current(Ens)=Ens'

<u>Entrée</u>
$Ens=\{o^1, o^2,\ldots,o^x\}$ est un ensemble d'objets entrepôt.

<u>Sortie</u>
$Ens'=\{S^1_0, S^2_0,\ldots,S^x_0\}$ est un ensemble d'états courant tel que $\forall i\in[1..x]$, S^i_0 représente l'état courant de chaque objet entrepôt o^i.

<u>*EXEMPLE :*</u> L'administrateur souhaite "*obtenir les noms des hôpitaux publics de Paris et la subvention courante qu'ils ont reçue*".

Project(x, **Current**(**Select**(xx, Etablissement, xx.statut="public" ∧
xx.adresse.ville="Paris")),
x.nom, x.subventions)

Le résultat obtenu est de type { [String, [Double, String, String, T_DomTemporel]] } et contient les valeurs suivantes :

{ ["Hôpital Hôtel Dieu", [2.000.000,00, "Ministère de la santé", "public",<[08-2000;now]>]],
["Hôpital du Val de Grâce", [20.000.000,00, "Ministère de la santé", "public"],<[01-2000;now]>] }

3.1.2 Opérateurs d'accès aux états passés : Past

<u>*Définition*</u> :
L'opérateur *Past* permet d'accéder aux états passés d'un ensemble d'objets entrepôt. La syntaxe de l'opérateur *Past* est la suivante :

Past(Ens)=Ens'

<u>Entrée</u>
$Ens=\{o^1,o^2,\ldots,o^x\}$ est un ensemble d'objets entrepôt.

<u>Sortie</u>
$Ens'=\{Ens^1,Ens^2,\ldots,Ens^x\}$ est un ensemble d'ensembles tel que $\forall i\in[1..x]$, $Ens^i=\{S^i_{p1}, S^i_{p2},\ldots,S^i_{pi}\}$ est l'ensemble des états passés de o^i.

<u>*EXEMPLE :*</u> Maintenant, l'administrateur souhaite "*obtenir l'historique détaillé des subventions perçues par les hôpitaux publics de Paris*".

Project(x, **Select**(xx, Etablissement, xx.statut="public" ∧
xx.adresse.ville="Paris"),
x.nom, **Project**(y, **Flatten**(**Past**({x})), y.subventions))

Le résultat obtenu est de type { [String, {[Double, String, String, T_DomTemporel]}] } et contient les valeurs suivantes :

{ ["Hôpital Hôtel Dieu",
{[10.000.000,00, "Conseil Régional Ile-de-France", "public",<[04-2000;07-2000]>],

[10.000.000,00, "Groupe Fabre", "privée",<[02-2000;03-2000]>],
[1.000.000,00, "Arc", "association",<[10-1999;01-2000]>],
[5.000.000,00, "Conseil Régional Ile-de-France", "public",<[06-1999;09-1999]>],
[10.000.000,00, "Groupe Fabre", "privée",<[03-1999;05-1999]>],
[1.000.000,00, "Arc", "association",<[01-1999;02-1999]>] }],
["Hôpital du Val de Grâce",
{[500.000,00, "Groupe fabre", "privée",<[04-2000;05-2000]>],
[,20.000.000,00, " Conseil Régional Ile-de-France ", "public",<[01-2000;03-2000]>],
[3.500.000,00, "Arc", "association",<[08-1999;12-1999]>],
[20.0000.000,00, "Ministère de la Santé", "public",<[04-1999;07-1999]>],
[18.000.000,00, "Ministère de la Santé", "public",<[01-1999;03-1999]>]}] }

3.1.3 Opérateurs d'accès aux états archivés : Archive

<u>*Définition*</u> :
L'opérateur *Archive* permet d'accéder aux états archivés d'un ensemble d'objets entrepôt. La syntaxe de l'opérateur *Archive* est la suivante :

Archive(Ens)=Ens'

<u>Entrée</u>
$Ens=\{o^1,o^2,\ldots,o^x\}$ est un ensemble d'objets entrepôt.

<u>Sortie</u>
$Ens'=\{Ens^1,Ens^2,\ldots,Ens^x\}$ est un ensemble d'ensembles tel que $\forall i\in[1..x]$, $Ens^i=\{S^i_{a1}, S^i_{a2},\ldots,S^i_{ai}\}$ est l'ensemble des états archivés de o^i.

<u>*EXEMPLE :*</u> L'administrateur souhaite "*obtenir les différentes évolutions du montant des subventions de 1995 à 1998 pour les hôpitaux publics de Paris* ".

Project(x, **Select**(xx, Etablissement, xx.statut="public" ∧
xx.adresse.ville="Paris"),
x.nom, **Project**(y, **Flatten**(**Archive**({x})), y.subventions))

Le résultat obtenu est de type { [String, {[Double, T_DomTemporel]}] } et contient les valeurs suivantes :

{ ["Hôpital Hôtel Dieu",
{[10.000.000,00, <[1997;1998]>],
[8.000.000,00, <[1995;1996]>] }],
["Hôpital du Val de Grâce",
{[15.000.000,00, <[1997;1998]>],
[10.000.000,00, <[1995;1996]>] }],

Opérateurs de déstructuration et de restructuration des objets entrepôt DisConnect et Connect

3.2 Opérateurs Connect et DisConnect

Les opérateurs précédents permettent d'atteindre les données courantes, passées et archivées en obtenant des ensembles d'états courants ou des ensembles d'ensembles d'états passés ou archivés. Nous proposons deux opérateurs supplémentaires qui permettent d'atteindre les états sans considérer leur sémantique (courant, passé, archivé).

- L'opérateur *DisConnect* permet d'obtenir tous les états des objets entrepôt sans tenir compte de la caractéristique d'états courants, passés ou archivés. L'intérêt de cet opérateur est de pouvoir observer facilement l'évolution d'un objet dans son ensemble.
- L'opérateur *Connect* permet d'obtenir des objets entrepôt à partir des ensembles d'états. Il s'agit de l'opération inverse de la précédente.

<u>*Définition*</u> :
L'opérateur *DisConnect* permet d'obtenir l'ensemble des états des objets entrepôt. La syntaxe de l'opérateur *DisConnect* est la suivante :

DisConnect(Ens)=Ens'

<u>Entrée</u>
$Ens=\{o^1, o^2,\ldots, o^x\}$ est un ensemble d'objets entrepôt tel que $\forall i\in[1..x]$, l'état courant de o^i est S^i_0, les états passés de o^i sont $\{S^i_{p1}, S^i_{p2},\ldots,S^i_{pi}\}$, les états archivés de o^i sont $\{S^i_{a1}, S^i_{a2},\ldots,S^i_{ai}\}$.

<u>Sortie</u>
$Ens'=\{Ens^1, Ens^2,\ldots, Ens^x\}$ est un ensemble d'ensembles tel que $\forall i\in[1..x]$, $Ens^i=\{S^i_0, S^i_{p1}, S^i_{p2},\ldots, S^i_{pi}, S^i_{a1}, S^i_{a2},\ldots, S^i_{ai}\}$ est l'ensemble d'états d'un objet entrepôt.

<u>*EXEMPLE :*</u> L'administrateur souhaite "*obtenir l'ensemble des évolutions du budget des différents services hospitaliers de l'hôpital Hôtel Dieu*".

DisConnect(**Select**(xx, Service, xx.etablissmt.nom="Hôpital Hôtel Dieu"))

Le résultat obtenu est de type { {[Double, T_DomTemporel]} } et contient les valeurs suivantes :

```
{ { [18.000.000,00, <[07-2000;now]>],      [8.000.000,00, <[01-2000;06-2000]>],
    [9.500.000,00, <[07-1999;12-1999]>],  [7.000.000,00, <[01-1999;06-1999]>],
    [6.500.000,00, <[01-1997;12-1998]>],  [6.000.000,00, <[01-1995;12-1996]>] },
  { [2.000.000,00, <[07-2000;now]>],       [2.400.000,00, <[01-2000;06-2000]>],
    [1.900.000,00, <[07-1999;12-1999]>],  [2.000.000,00, <[01-1999;06-1999]>],
    [2.500.000,00, <[01-1997;12-1998]>],  [2.000.000,00, <[01-1995;12-1996]>] },
  { [15.000.000,00, <[07-2000;now]>],      [10.000.000,00, <[01-1999;02-1999]>],
    [12.000.000,00, <[03-1999;06-1999];[10-1999;12-1999]>],
    [14.000.000,00, <[07-1999;06-1999];[01-2000;02-2000]>],
    [16.000.000,00, <[03-2000;06-2000]>],
    [17.500.000,00, <[01-1997;12-1998]>],
    [15.000.000,00, <[01-1995;12-1996]>] } }
```

<u>*Définition*</u> :
L'opérateur *Connect* permet de reconstruire un ensemble d'objets entrepôt à partir de l'ensemble (ou d'un sous ensemble) d'états. La syntaxe de l'opérateur *Connect* est la suivante :

Connect(Ens)=Ens'

<u>Entrée</u>
$Ens=\{Ens^1, Ens^2,\ldots, Ens^x\}$ est un ensemble d'ensembles tel que $\forall i\in[1..x]$, $Ens^i=\{S^i_0, S^i_{p1}, S^i_{p2},\ldots, S^i_{pi}, S^i_{a1}, S^i_{a2},\ldots, S^i_{ai}\}$ est un ensemble des états d'un objet entrepôt.

<u>Sortie</u>
$Ens'=\{o^1, o^2,\ldots, o^x\}$ est un ensemble d'objets entrepôt reconstitués tel que $\forall i\in[1..x]$, o^i est reconstitué avec les états de Ens^i.

Remarque :
L'opérateur *Connect* fait appel à des méta-données afin de reconstruire les objets entrepôt. Il s'agit d'une opération complexe qui permet de retrouver la structure des objets entrepôt à partir d'un ensemble d'ensemble de leurs états. Pour cela, le système doit être capable de conserver et de restituer la structuration de chaque objet entrepôt.

La Figure 31 illustre le fonctionnement des opérateurs *DisConnect* et *Connect*.

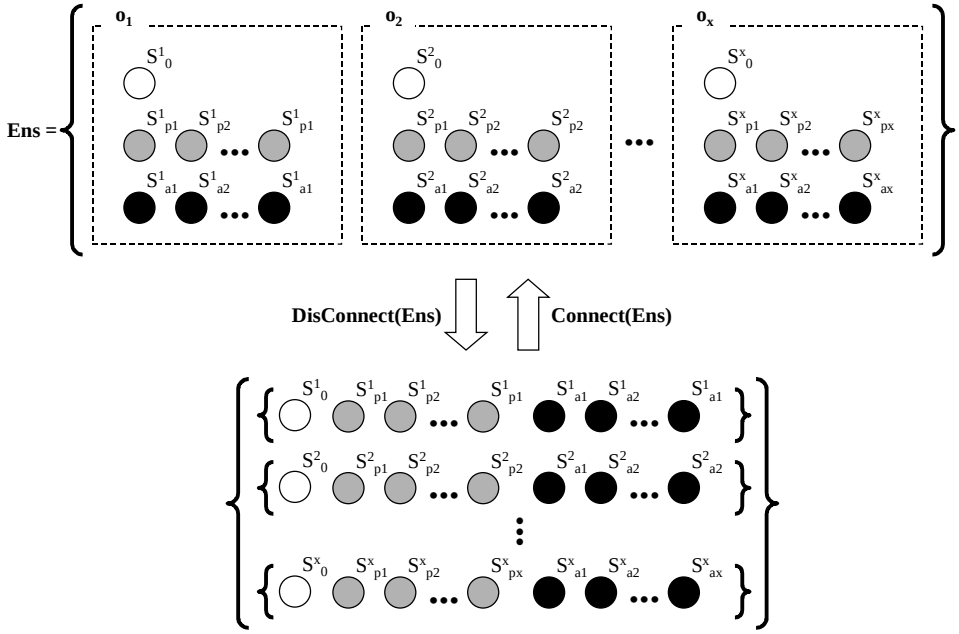


Figure 31 : Principes des opérateurs *DisConnect* et *Connect*.

3.3 Synthèse

Nous avons défini un ensemble d'opérateurs spécifiques permettant de manipuler des ensembles d'objets entrepôt et d'états.

- Nous avons introduit trois opérateurs spécifiques (*Current*, *Past*, *Archive*) permettant d'atteindre les différents états ou ensembles d'états composants les objets entrepôt.
- Nous avons défini un opérateur (*DisConnect*) permettant d'obtenir simplement l'ensemble des états des objets entrepôt, puis nous avons proposé un nouvel opérateur (*Connect*) qui permet, en s'appuyant sur les méta-données, de reconstruire des objets entrepôt partir d'un ensemble de leurs états indépendants.

4 MANIPULATION DU DOMAINE TEMPOREL DES ETATS

Un objet entrepôt est constitué d'états contenant la valeur de l'objet à des instants précis, modélisés par un domaine temporel. Il est nécessaire d'offrir des mécanismes simples de manipulation du temps afin d'exploiter notre modélisation de l'évolution au niveau de l'interrogation.

Nous rappelons que le domaine temporel associé aux états est modélisé par un ensemble ordonné d'intervalles non vides, disjoints et non contigus, noté $DomT^i_j=<Int_1; Int_2;\ldots; Int_{hj}>$ où $\forall k\in[1..h]$, $Int_k=[TDeb_k; TFin_k]$.

4.1 Généralisation des relations de Allen

Afin de faciliter la manipulation des domaines temporels, nous généralisons les treize relations définies par Allen [Allen 1983] aux domaines temporels associés aux états des objets entrepôt. L'intérêt de cette généralisation est qu'elle offre la puissance nécessaire pour combiner simplement les domaines temporels, c'est à dire les valeurs temporelles associées aux états des objets entrepôt.

Soient X et Y deux domaines temporels. On pose

- $X=<[TDeb^X_1, TFin^X_1], [TDeb^X_2, TFin^X_2],\ldots, [TDeb^X_{hx}, TFin^X_{hx}]>$ et
- $Y=<[TDeb^Y_1, Tfin^Y_1], [TDeb^Y_2, Tfin^Y_2],\ldots, [TDeb^Y_{hy}, Tfin^Y_{hy}]>$.

Le Tableau 18 étend les treize relations de Allen entre ces domaines temporels.

Relation	Définition	Réciproque
X Precedes Y	$TFin^X_{hx}<TDeb^Y_1$	*Y Follows X*
X Meets Y	$TFin^X_{hx}=TDeb^Y_1$	*Y IsMeeted X*
X Overlaps Y	$\exists Int_i\in X, \exists Int_j\in Y, TDeb^X_i<TDeb^Y_j \wedge TDeb^Y_j<TFin^X_i \wedge TFin^X_i<TFin^Y_j$	*Y IsOverlaped X*
X During Y	$\forall Int_i\in X, \exists Int_j\in Y, TDeb^X_i>TDeb^Y_j \wedge TFin^X_i<TFin^Y_j$	*Y IsDuringX*
X Starts Y	$TDeb^X_1=TDeb^Y_1$	*Y IsStarted X*
X Ends Y	$TFin^X_{hx}=TFin^Y_{hy}$	*Y IsFinished X*
X Equals Y	$\forall i\in[1,h_1], h_1=h_2, TDeb^X_i=TDeb^Y_i \wedge TFin^X_i=TFin^Y_i$	*Y Equals X*

Tableau 18 : Relations de Allen étendues aux domaines temporels.

Remarque :
L'extension des relations de Allen peut être complétée au travers de relations définies dans [Ladkin 1987] entre intervalles non convexes. Peter Ladkin propose de caractériser les relations de Allen (R) par des extensions comme R *toujours*, R *parfois*, R *la plupart du temps*,… Nous ne détaillons pas ces extensions dans cette thèse, néanmoins il est possible de les intégrer afin d'améliorer l'expressivité du langage.

Remarque :
Les instants et les intervalles peuvent s'exprimer facilement comme un cas particulier des domaines temporels. Ceci permet d'appliquer les relations de Allen entre ces différents types temporels.

4.2 Opérateur de restriction temporelle : State

Lors de la manipulation directe des données de l'entrepôt, il est souvent nécessaire d'obtenir une information à une date particulière. Autrement dit, un administrateur doit pouvoir obtenir l'état d'un ensemble d'objets entrepôt en fonction d'une valeur temporelle (modélisée soit par un instant précis, soit par un intervalle de temps, soit par un domaine temporel).

Définition :
L'opérateur *State* permet d'obtenir les états d'un ensemble d'objets entrepôt en fonction d'une valeur temporelle (d'un instant, d'un intervalle, d'un domaine temporel). La syntaxe de l'opérateur *State* est la suivante :

State(Ens, T, op_{al})=Ens'

Entrées
$Ens=\{o^1, o^2,\ldots,o^x\}$ est un ensemble d'objets entrepôt,

T est un élément temporel pouvant être un domaine temporel, un intervalle ou un instant,

op_{al} est un opérateur de Allen.

Sortie
Ens est un ensemble d'ensembles d'états, $Ens=\{\{S^i_j\} \mid S^i_j \in o^i \wedge T \text{ op}_{\text{al}}\ DomT^i_j\}$.

Les états qui constituent les objets entrepôt ont des types différents ; en effet, pour un type d'objets entrepôt le type des états courants est sous-type des états passés, lui-même sous-type des états archivés. Par conséquent, le type des états de *Ens'* est réduit aux propriétés communes de tous les états :

- si les objets entrepôt contiennent des états archivés, le type des états de l'ensemble résultat est le même que les états archivés (on notera $S|_a$ un état S réduit aux propriétés des états archivés) ;
- si les objets entrepôt contiennent des états temporels et pas d'états archivés, le type des états de l'ensemble résultat est le même que les états temporels (on notera $S|_p$ un état S réduit aux propriétés des états passés) ;
- si les objets entrepôt contiennent que des états courants, le type des états de l'ensemble résultat est le même que les états courants.

EXEMPLE : L'administrateur souhaite "*obtenir le montant de la subvention de l'hôpital Hôtel Dieu en juin 2000*".

State(**Flatten**(**Select**(xx1, Etablissement, xx1.nom="Hôpital Hôtel Dieu")),
T_Instant(06-2000, mois), During)

Le résultat est de type { [[Double, String, String], T_DomTemporel] } et contient les valeurs suivantes :

{ [[10.000.000,00, "Conseil-Régional Ile-de-France", "public"],
<[04-2000;07-2000]>] }

L'administrateur souhaite "*obtenir les montants des subventions de l'hôpital Hôtel Dieu entre janvier 2000 et juin 2000*".

State(**Flatten**(**Select**(xx1, Etablissement, xx1.nom="Hôpital Hôtel Dieu")),
T_Interval(01-2000, 06-2000, mois), Overlaps)
Union
State(**Flatten**(**Select**(xx1, Etablissement, xx1.nom="Hôpital Hôtel Dieu")),
T_Interval(01-2000, 06-2000, mois), IsDuring)

Le résultat est de type { [[Double, String, String], T_DomTemporel] } et contient les valeurs suivantes :

```
{ [[10.000.000,00, "Conseil-Régional Ile-de-France", "public"],
   <[04-2000;07-2000]>],
  [[10.000.000,00, "Groupe Fabre", "privée"],
   <[02-2000;03-2000]>],
  [[1.000.000,00, "Arc", "association"],
   <[10-1999;01-2000]>] }
```

4.3 Opérateurs de jointures temporelles : UJoin et IJoin

La jointure est étendue pour permettre de combiner les ensembles d'états. Cette adaptation de la jointure concerne le caractère temporel des états pour lequel deux options sont possibles pour calculer le résultat :

- le résultat est calculé à partir des valeurs de chaque ensemble, aux instants en commun (intersection des domaines temporels des états) ;
- le résultat est calculé à partir des valeurs de chaque ensemble, pour les instants de chacun (union des domaines temporels des états).

Définition :
La jointure par intersection des domaines temporels entre deux ensembles d'états est décrit de la manière suivante :

$$\text{IJoin}(x_1, \text{Ens}^1, x_2, \text{Ens}^2, P)=\text{Ens}$$

Entrées
Ens^1 et Ens^2 sont deux ensembles d'états de même type et dont les domaines temporels sont de même unité temporelle,

x_1 et x_2 sont les variables d'itération sur les éléments des ensembles désignés Ens^1 et Ens^2,

P est un prédicat de jointure.

Sortie
Ens est l'ensemble des n-uplets résultat tel que $Ens=\{(DomT, V_1, V_2) \mid \exists S_1 \in Ens^1, \exists S_2 \in Ens^2, DomT=DomT_1 \cap DomT_2 \wedge V_1 \in S_1 \wedge V_2 \in S_2 \wedge P(V_1, V_2)\}$.

EXEMPLE : L'administrateur souhaite "*obtenir les subventions lorsque le montant de celle de l'hôpital Hôtel Dieu était supérieure à celle de l'hôpital du Val de Grâce pour l'historique détaillé des deux établissements*".

Si l'administrateur utilise la jointure par intersection des domaines temporels, il exprime la requête suivante :

```
IJoin(h1,
    Flatten(Past(Select(xx1, Etablissement, xx1.nom="Hôpital Hôtel Dieu"))),
    h2,
    Flatten(Past(Select(xx2, Etablissement, xx2.nom="Hôpital du Val de Grâce"))),
     h1.montant > h2.montant)
```

Le résultat est de type { [[Double, String, String], [Double, String, String], T_DomTemporel] } et contient les valeurs suivantes :

{ [[10.000.000,00, "Groupe Fabre", "privée"],
[3.000.000,00, "Asso. maladies orphelines", "association"],
<[05-1999;05-1999]>],

[[5.000.000,00, "Conseil Régional Ile-de-France", "public"],
[3.000.000,00, "Asso. maladies orphelines", "association"],
<[06-1999;07-1999]>] }

Définition :
La jointure par union des domaines temporels entre deux ensembles d'états est décrit de la manière suivante :

$$\text{UJoin}(x_1, \text{Ens}^1, x_2, \text{Ens}^2, P)=\text{Ens}$$

Entrées
Ens^1 et Ens^2 sont deux ensembles d'états de même type et dont les domaines temporels sont de même unité temporelle,

x_1 et x_2 sont les variables d'itération sur les éléments des ensembles désignés Ens^1 et Ens^2,

P est un prédicat de jointure.

Sortie
Ens est l'ensemble des n-uplets résultat tel que $Ens=\{(DomT, V_1, V_2) \mid \exists S_1 \in Ens^1, \exists S_2 \in Ens^2, V_1 \in S_1 \wedge V_2 \in S_2 \wedge P(V_1, V_2) \wedge ((DomT \subseteq DomT_2 \wedge V=V_1 \wedge (DomT, V) \in Ens^1) \vee (DomT \subseteq DomT_1 \wedge V=V_2 \wedge (DomT, V) \in Ens^2))\}$.

EXEMPLE : On reprend l'exemple précédent où l'administrateur souhaite "obtenir les subventions lorsque le montant de celle de l'hôpital Hôtel Dieu était supérieure à celle de l'hôpital du Val de Grâce pour l'historique détaillé des deux établissements".

Si l'administrateur utilise la jointure par union des domaines temporels, il exprime la requête suivante :

UJoin(h1,
Flatten(**Past**(**Select**(xx1, Etablissement, xx1.nom="Hôpital Hôtel Dieu"))),
h2,
Flatten(**Past**(**Select**(xx2, Etablissement, xx2.nom="Hôpital du Val de Grâce"),
h1.montant > h2.montant)

Le résultat est de type { [[Double, String, String], [Double, String, String], T_DomTemporel] } et contient les valeurs suivantes :

{ [[10.000.000,00, "Groupe Fabre", "privée"],
[3.000.000,00, "Asso. maladies orphelines", "association"],
<[03-1999;07-1999]>],

[[5.000.000,00, "Conseil Régional Ile-de-France", "public"],
[3.000.000,00, "Asso. maladies orphelines", "association"],
<[05-1999;09-1999]>] }

La différence réside dans la construction des domaines temporels : la première requête effectue l'intersection des domaines temporels des couples de valeurs satisfaisants la jointure tandis que le seconde requête effectue l'union des domaines temporels.

- Dans le cas de la jointure par intersection, l'administrateur obtient les dates durant lesquelles la condition a été vérifiée pour tous les instants.
- Dans le cas de la jointure par union, l'administrateur obtient les dates durant lesquelles la condition a été vérifiée au moins à un instant.

4.4 Opérateurs de restructuration temporelle : UGroup et DGroup

L'aspect temporel des objets entrepôt permet l'introduction de deux nouveaux opérateurs réalisant des regroupements en fonction de critères temporels :

- un opérateur de groupement se base sur les unités temporelles en permettant de regrouper un ensemble d'états à une unité temporelle supérieure ;
- un opérateur de groupement se base sur la durée en permettant de regrouper un ensemble d'états par durées successives depuis le temps d'origine de l'ensemble des états.

Définition :
L'opérateur *UGroup* permet de grouper un ensemble d'états selon une unité temporelle supérieure. La syntaxe de l'opérateur *UGroup* est la suivante :

UGroup(Ens, U)=Ens'

Entrées
$Ens=\{S_1, S_2,\ldots, S_n\}$ est un ensemble d'états dont l'unité temporelle est inférieure à U, $\forall i \in [1..n]$, $Unit(S_i)<U$,

U désigne une unité temporelle.

Sortie
Ens' est un ensemble de n-uplets, dont la valeur est l'ensemble des valeurs structurelles des états *Ens*, regroupés à unité temporelle *U*.

EXEMPLE : "*Obtenir l'historique détaillé des subventions de la clinique du Sport, regroupées par années*"
U_Group(**Flatten**(**Past**(**Select**(e, Etablissement, e.nom="Clinique du Sport"))), annee)

Le résultat obtenu est de type { [{[Double, String, String]} ; T_DomTemporel] } et contient les valeurs suivantes :
{ [{ [18.000.000,00; "Ministère de la Santé" ; "public"] ;
[20.000.000,00; "Ministère de la Santé" ; "public"] ;
[3.500.000,00; "Arc" ; "association"] } ; <[1999; 1999]>] ;
[{ [20.000.000,00; "Conseil Régional Ile-de-France" ; "public"] ;
[500.000,00; "Groupe Fabre" ; "privée"] ; } ; <[2000; 2000]>] }

Définition :
L'opérateur *DGroup* permet de grouper un ensemble d'états selon une durée depuis l'instant minimum des états de l'ensemble de départ. La syntaxe de l'opérateur *DGroup* est la suivante :

DGroup(Ens, D)=Ens'

Entrées
Ens={S_1, S_2,…, S_n} est un ensemble d'états,

D désigne une durée de même unité temporelle que les états.

Sortie
Ens' est un ensemble de n-uplets, dont chaque valeur est l'ensemble des valeurs structurelles des états *Ens*, regroupés par période de temps (exprimée au travers d'une durée depuis le plus petit des instants des domaines temporels des états).

EXEMPLE : "*Obtenir l'historique détaillé des subventions de la clinique du Sport, regroupées par périodes de quatre mois*"

D_Group(**Flatten**(**Past**(**Select**(e, Etablissement, e.nom="Clinique du Sport"))),
T_Duration(mois, 4))

Le résultat obtenu est de type { [{[Double, String, String]} ; T_DomTemporel] } et contient les valeurs suivantes :

{ [{[18.000.000,00; "Ministère de la Santé" ; "public"] ;
[20.000.000,00; "Ministère de la Santé" ; "public"]}; <01-1999;04-1999>] ;
[{[20.000.000,00; "Ministère de la Santé" ; "public"] ;
[3.500.000,00; "Arc" ; "association"]}; <05-1999;08-1999>] ;
[{[3.500.000,00; "Arc" ; "association"]}; <09-1999;12-1999>] ;
[{[20.000.000,00; "Conseil Régional Ile-de-France" ; "public"] ;
[500.000,00; "Groupe Fabre" ; "privée"]}; <01-2000;04-2000>] ;
[{[500.000,00; "Groupe Fabre" ; "privée"]}; <05-1999;08-2000>] ;

On constate que certaines valeurs sont répétées dans plusieurs ensembles de regroupement. Cela résulte du fait que le domaine temporel de l'état (dont la valeur dupliquée est obtenue) chevauche le domaine temporel de groupement.

Remarque :
La valeur structurelle de l'état est placée dans l'ensemble de regroupement dont le domaine temporel contient celui de l'état, mais également dans chaque ensemble de regroupement dont le domaine temporel chevauche celui de l'état.

4.5 Synthèse

Nous avons adapté les opérateurs de Allen aux domaines temporels (qui datent les valeurs structurelles enregistrées dans les états). Puis, nous avons proposé un opérateur (*State*) qui permet d'obtenir les états des objets entrepôt en fonction d'une valeur temporelle combinée une relation de Allen étendue. Cet opérateur permet à l'utilisateur de retrouver l'état (ou les états) des objets entrepôt en fonction de valeurs temporelles.

Nous avons adopté également des opérateurs permettant de réaliser des groupements selon des critères temporels (*UGroup*, *DGroup*) et nous avons adapté l'opération de jointure à la dimension temporelle en introduisant des jointures temporelles (*UJoin*, *IJoin*).

5 MANIPULATION DES SERIES D'ETATS

La prise de décision s'appuie très souvent sur les évolutions organisées chronologiquement [Chaudhuri, Dayal 1997] ; il s'agit de manipuler ces chronologies en appliquant systématiquement un traitement sur chaque valeur. Or, les opérateurs précédents ne permettent pas d'effectuer ce type de manipulation.

Considérons l'objet o^3 représentant le service des urgences de l'hôpital Hôtel Dieu à Paris. Son historique détaillé (l'ensemble des états passés) comprend quatre états passés dont les domaines temporels s'entrecroisent. Or les analyses (notamment celles réalisées en statistique) utilisent couramment des traitements inapplicables sur de tels domaines temporels. Par exemple il est souvent utile d'appliquer une agrégation par accumulation sur des états (qui doivent être organisés chronologiquement) pour obtenir de nouveaux états dont la valeur est calculée en agrégeant les valeurs cumulées des états initiaux. De tels traitements n'ont de sens que sur une série d'états chronologiquement ordonnés.

Cette section se propose d'introduire un opérateur de transformation des objets entrepôt en organisant chronologiquement les états qui les composent (on parlera de série temporelle [Teste 1999a]), ceci afin de faciliter l'analyse et les traitements des évolutions de données. Nous définissons plusieurs opérateurs dédiés aux séries temporelles d'états.

5.1 Opérateur de formatage MakeSerie

Nous définissons une série temporelle d'états à partir d'un ensemble d'états dont les domaines temporels sont réorganisés sous la forme d'un intervalle. L'ensemble des états est ordonné chronologiquement.

<u>*Définition*</u> :
Une série temporelle d'états est un ensemble ordonné d'états $<S_1, S_2,\ldots, S_s>$ tel que

- les domaines temporels des états sont des intervalles, $\forall i\in[1..s]$, $domT_i=<[TDeb_i;TFin_i]>$,
- les états sont ordonnés chronologiquement, $\forall i\in[1..s-1]$, $domT_i$ Precedes $domT_{i+1}$.

Nous proposons un nouvel opérateur de transformation des objets entrepôt en série temporelle. Il s'agit d'un formatage des données sous la forme d'une série temporelle d'états.

<u>*Définition*</u> :
L'opérateur *MakeSerie* transforme un ensemble d'états en une série temporelle d'états. Il se définit de la manière suivante :

MakeSerie(Ens)=Lst

<u>Entrée</u>
$Ens=\{S_1, S_2, S_n\}$ est un ensemble d'états dont les domaines temporels ne se chevauchent pas ; $\forall i\in[1..n]$, $\forall j\in[1..n]$, $i\neq j$, $\neg$(*DomTi* overlaps *DomTj*).

<u>Sortie</u>
$Lst=<S_{s1}, S_{s2},\ldots, S_{ss}>$ est une série temporelle d'états.

Un objectif de cette transformation est de rendre plus simple la compréhension de l'évolution dans le temps d'un objet entrepôt.

EXEMPLE : L'administrateur souhaite créer une série temporelle des valeurs passées du budget pour le service d'urgence de l'hôpital Hôtel Dieu de Paris.

MakeSerie(**Flatten**(**Past**(**Select**(s, Services, s.nom="urgences" &
s.etablismt.nom="Hôpital Hôtel Dieu"))))

Le résultat obtenu est de type < [Double, T_DomTemporel] > et contient les valeurs suivantes :
< [10.000.000,00; <[01-1999;02-1999]>] ; [12.000.000,00; <[03-1999;06-1999]>] ;
[14.000.000,00; <[07-1999;09-1999]>] ; [12.000.000,00; <[10-1999;12-1999]>] ;
[14.000.000,00; <[01-2000;02-2000]>] ; [16.000.000,00; <[03-2000;06-2000]>] >

On notera SR le résultat obtenu pour compléter la requête dans les autres exemples de la section.

5.2 Transformations des séries temporelles

Nous adoptons différentes transformations pour les séries temporelles d'états :

- une opération d'agrégation (*Agreg*),
- une opération d'agrégation cumulée (*ACum*),
- une opération d'agrégation dynamique (*AMove*).

5.2.1 Opérateur Agreg

Les ensembles d'états peuvent être agrégés au moyen d'un opérateur d'agrégation. Il permet de visualiser les données à un niveau de détail plus synthétique.

Définition :
L'opérateur d'agrégation *Agreg* transforme une série temporelle d'états en une valeur. Il se définit de la manière suivante :

Agreg(SR, Archi)=V

Entrées
$SR=<S_{s1}, S_{s2},\ldots, S_{ss}>$ est une série temporelle d'états,

$Archi=\{(att_1, f_1), (att_2, f_2),\ldots, (att_t, f_t)\}$ est un filtre d'archivage, c'est à dire un ensemble d'attributs (des états de *SR*) associés à une fonction d'agrégation.

Sortie
V est la valeur résultat de l'agrégation des valeurs structurelles des états de *Ens*.

EXEMPLE : "*Obtenir la moyenne du budget du service d'urgence de l'hôpital Hôtel Dieu pour l'ensemble de ses états passés*"
Agreg(SR, {(budget, avg)})

5.2.2 Opérateur ACum

L'opérateur d'agrégation cumulative consiste à cumuler les résultats d'une agrégation appliquée aux valeurs successives.

Définition :
L'opérateur d'agrégation cumulative *ACum* transforme une série temporelle d'états en une autre série d'états dont les valeurs sont le résultat d'une agrégation cumulée. L'opérateur *ACum* se définit de la manière suivante :

ACum(SR, Archi)=SR'

Entrées
$SR=<S_{s1}, S_{s2},\ldots, S_{ss}>$ est une série temporelle d'états,

$Archi=\{(att_1, f_1), (att_2, f_2),\ldots, (att_t, f_t)\}$ est un filtre d'archivage, c'est à dire un ensemble d'attributs (des états de *SR*) associés à une fonction d'agrégation.

Sortie
$SR'=<S_{a1}, S_{a2},\ldots, S_{aa}>$ est une série temporelle d'états dont les valeurs structurelles sont le résultat d'une agrégation cumulée.

EXEMPLE : "Observer l'évolution du service d'urgence de l'hôpital Hôtel Dieu en effectuant la somme cumulée du budget pour l'ensemble de ses états passés "
ACum(SR, {(Budget, sum)})

Le résultat obtenu est de type < [Double, T_DomTemporel] > et contient les valeurs suivantes :
< [10.000.000,00; <[01-1999;02-1999]>] ;
[22.000.000,00; <[01-1999;06-1999]>] ;
[36.000.000,00; <[01-1999;09-1999]>] ;
[48.000.000,00; <[01-1999;12-1999]>] ;
[62.000.000,00; <[01-1999;02-2000]>] ;
[78.000.000,00; <[01-1999;06-2000]>] >

5.2.3 Opérateur AMove

L'opérateur d'agrégation dynamique permet d'appliquer des agrégations à des ensembles d'états sur une période de temps "*qui se déplace*". Cet opérateur consiste à cumuler les résultats d'une agrégation appliquée aux valeurs des états appartenant à un intervalle, puis à recommencer en décalant l'intervalle.

Définition :
Les opérateurs d'agrégation cumulative dynamique *AMove* transforment un ensemble d'états en un autre ensemble d'états dont les valeurs sont le résultat d'une agrégation cumulée par glissements. Les opérateurs *AMove* se définissent de la manière suivante :

AMove(SR, Archi, D)=SR'

Entrées
$SR=<S_{s1}, S_{s2},\ldots, S_{ss}>$ est une série temporelle d'états,

$Archi=\{(att_1, f_1), (att_2, f_2),\ldots, (att_t, f_t)\}$ est un filtre d'archivage, c'est à dire un ensemble d'attributs (des états de *SR*) associés à une fonction d'agrégation,

D est une durée de même unité temporelle que les états de *Ens*.

Sortie
$SR=<S_{a1}, S_{a2},\ldots, S_{aa}>$ est une série temporel d'états dont les valeurs structurelles sont le résultat d'une agrégation cumulée.

EXEMPLE : "*Observer l'évolution du service d'urgence de l'hôpital Hôtel Dieu en effectuant la somme cumulée du budget par semestre pour l'ensemble de ses états passés* "
AMove(SR, {(Budget, sum)}, T_Duree(mois, 6))

Le résultat obtenu est de type < [Double, T_DomTemporel] > et contient les valeurs suivantes :
< [22.000.000,00; <[01-1999;06-1999]>] ;
[26.000.000,00; <[07-1999;12-1999]>] ;
[30.000.000,00; <[01-2000;06-2000]>] >

5.3 Changement de l'échelle d'observation

Un traitement couramment utilisé lors des analyses est celui du changement de l'échelle d'observation, c'est à dire celui du changement de la granularité des domaines temporels des états. Cette transformation peut s'opérer dans deux sens.

- Soit la granularité est augmentée, c'est à dire que l'on augmente l'unité temporelle des domaines temporels (cela revient à réduire le détail d'observation des évolutions). On parle de transformation "*scale-up*".
- Soit la granularité est diminuée, c'est à dire que l'on diminue l'unité temporelle des domaines temporels (cela revient à augmenter le détail d'observation des évolutions). On parle de transformation "*scale-down*".

5.3.1 Opérateur ScaleUp

Définition :
L'opérateur *ScaleUp* transforme un ensemble d'états en augmentant son unité temporelle. L'opérateur *ScaleUp* se définit de la manière suivante :

ScaleUp(SR, U^+, Archi)=SR'

Entrées
$SR=<S_{s1}, S_{s2},\ldots, S_{ss}>$ est une série temporelle d'états (d'unité temporelle U_s),

U^+ est une unité temporelle telle que $U_s \leq U^+$,

$Archi=\{(att_1, f_1), (att_2, f_2),\ldots, (att_t, f_t)\}$ est un filtre d'archivage, c'est à dire un ensemble d'attributs (des états de *SR*) associés à une fonction d'agrégation (*sum*, *count*, *avg*, *max*, *min*…).

Sortie
$SR'=<S_{a1}, S_{a2},\ldots, S_{aa}>$ est une série temporel d'états (d'unité temporelle U^+). Chaque état S_{ai} est obtenu en appliquant le filtre d'archivage sur les valeurs structurelles des états de *Ens* dont le domaine temporel est contenu ou chevauche le domaine temporel de S_{ai}.

EXEMPLE : "*Observer les états passés du service d'urgence de l'hôpital Hôtel Dieu par année*"
ScaleUp(SR, annee, {(Budget, avg)})=SR^+

Le résultat obtenu est de type < [Double, T_DomTemporel] > et contient les valeurs suivantes :
< [12.000.000,00; <[1999;1999]>] ; [15.000.000,00; <[2000;2000]>] >

Le changement d'échelle permet d'observer les évolutions détaillées de l'hôpital Hôtel Dieu en passant de l'unité temporelle mois à celle de année. Pour cela, les états sont transformés en série temporelle (suppression des entrecroisement de domaines temporels).

Les états sont regroupés par année. Lorsque le domaine temporel d'un état chevauche deux années, l'état est pris en compte sur les deux années.

L'agrégation est appliquée ; sur cet exemple, la moyenne du budget des états est calculée par rapport au nombre d'états (de la série temporelle).

5.3.2 Opérateur ScaleDown

L'opération *ScaleDown* annule l'effet de *ScaleUp* : on transforme les états en diminuant l'unité temporelle. Cette opération utilise des méta-données car elle nécessite de conserver l'information détaillée de chaque état. En outre, il n'est pas possible de diminuer l'unité d'observation au delà de l'unité initiale des états U_s.

Définition :
L'opérateur *ScaleDown* transforme un ensemble d'états en diminuant l'unité temporelle. L'opérateur *ScaleDown* se définit de la manière suivante :

ScaleDown(SR, U^-, Archi)=SR'

Entrée
$SR=<S_{s1}, S_{s2},\ldots, S_{ss}>$ est une série temporelle d'états (d'unité temporelle U^+ et d'unité temporelle d'origine U_s),

U^- est une unité temporelle telle que $U_s \leq U^- \leq U^+$,

$Archi=\{(att_1, f_1), (att_2, f_2),\ldots, (att_t, f_t)\}$ est un filtre d'archivage, c'est à dire un ensemble d'attributs (des états de *Ens*) associés à une fonction d'agrégation (*sum*, *count*, *avg*, *max*, *min*…).

Sortie
$SR'=<S_{a1}, S_{a2},\ldots, S_{aa}>$ est une série temporel d'états (d'unité temporelle U^-). Chaque état S_{ai} est obtenu en appliquant le filtre d'archivage sur les valeurs structurelles des états de *Ens* dont le domaine temporel est contenu ou chevauche le domaine temporel de S_{ai}.

EXEMPLE : On poursuit l'exemple précédent. A partir de SR^+, on souhaite "*Observer les états passés du service d'urgence de l'hôpital Hôtel Dieu par trimestre*"
ScaleDown(SR^+, trimestre, {(Budget, avg)})

Le résultat obtenu est de type < [Double, T_DomTemporel] > et contient les valeurs suivantes :
< [11.000.000,00;<[1-1999]>] ; [12.000.000,00;<[2-1999]>] ;
[14.000.000,00;<[3-1999]>] ; [12.000.000,00;<[4-1999]>] ;
[15.000.000,00;<[1-2000]>] ; [16.000.000,00;<[2-2000]>] >

L'intérêt de ces opérateurs est qu'ils transforment les évolutions suivant des niveaux de détails variables. Ceci permet aux utilisateurs d'observer les évolutions avec plus ou moins de finesse. Ils peuvent mieux appréhender les évolutions, en partant d'un faible niveau de détail (grande unité temporelle) vers des niveaux de détail inférieurs. En outre, l'opérateur de restriction des états (*State*) permet de se focaliser sur un sous-ensemble de ses états.

5.4 Synthèse

Nous avons défini un mécanisme de transformation des ensembles d'états (*MakeSerie*) permettant de formater les évolutions des données. Cette transformation en série temporelle

permet différentes opérations de manipulation : agréger les données (*Agreg*), transformer la série d'états par agrégations cumulées (*ACum*), transformer la série d'états par agrégations dynamiques (*AMove*) et transformer la série d'états en changeant l'unité temporelle (l'échelle du temps) avec laquelle les données sont observées (*ScaleUp*, *ScaleDown*).

6 CONCLUSION

Ce chapitre présente une algèbre permettant la manipulation des données de l'entrepôt. Notre proposition s'inscrit dans le cadre des algèbres objet et intègre les caractéristiques inhérentes à notre modèle d'entrepôts (objets entrepôt, états, domaines temporels). Nous avons défini plusieurs opérateurs.

Opérateur	Entrées	Sortie
V_Union, V_Intersection, V_Difference	{T_ObjetED}x{T_ObjetED}	{T_ObjetED}
	{T_Etat}x{T_Etat}xT_egalite	{T_Etat}
I_Union, I_Intersection, I_Difference	{T_ObjetED}x{T_ObjetED}	{T_ObjetED}
Flatten	{{T_ObjetED}}	{T_ObjetED}
	{{T_Etat}}	{T_Etat}
DupEliminante	{T_ObjetED}	{T_ObjetED}
	{T_Etat}	{T_Etat}
EmptyEliminate	{{T_ObjetED}}	{{T_ObjetED}}
	{{T_Etat}}	{{T_Etat}}
Project	T_iterateurx{T_ObjetED}xT_cheminx…xT_chemin	{T_Valeur}
	T_iterateurx{T_Etat}xT_cheminx…xT_chemin	{T_Valeur}
Select	{T_ObjetED}xT_predicat	{T_ObjetED}
	{T_Etat}xT_predicat	{T_Etat}
Join	{T_ObjetED}x{T_ObjetED}xT_predicat	{T_ObjetED}
	{T_Etat}x{T_Etat}xT_predicat	{T_Etat}
Nest, UnNest	{T_ObjetED}xString	{T_ObjetED}
	{T_Etat}xString	{T_Etat}
State	{T_ObjetED}xT_relationxT_temporel	{{T_Etat}}
UJoin, IJoin	{T_Etat}x{T_Etat}xT_predicat	{T_Etat}
UGroup	{T_Etat}xT_DomTemporel	{T_Valeur}
DGroup	{T_Etat}xT_Duree	{T_Valeur}
Current	{T_ObjetED}	{T_Etat}
Past, Archive	{T_ObjetED}	{{T_Etat}}
DisConnect	{T_ObjetED}	{T_Etat}
Connect	{T_Etat}	{T_ObjetED}
MakeSerie	{T_Etat}	<T_Etat>
Agreg	{T_Etat}xT_FiltreArchi	T_Valeur
ACum	<T_Etat>xT_FiltreArchi	<T_Etat>
AMove	<T_Etat>xT_FiltreArchixT_Duree	<T_Etat>
ScaleUp, ScaleDown	<T_Etat>xT_FiltreArchixT_Unit	<T_Etat>

Tableau 19 : Description des opérateurs pour la manipulation de l'entrepôt.

Le Tableau 19 décrit l'ensemble des opérateurs présentés ; pour chacun, le type des paramètres en entrée et le type du résultat en sortie sont précisés.

Les opérateurs ensemblistes confèrent à notre algèbre la puissance de l'approche ensembliste. Nous avons retenu les principaux opérateurs (V_Union, V_Difference, V_Intersection, I_Union, I_Difference, I_Intersection, Flatten, DupEliminate, EmptyEliminate). Les principaux opérateurs des algèbres pour objets (Select, Project, Join, Nest, UnNest) sont adaptés à notre modélisation des entrepôts.

L'ensemble des opérateurs de manipulation des objets entrepôt et des états est étendu par des opérateurs de manipulation intégrant l'aspect temporel des états (UJoin, IJoin, U_Group, D_Group). Des opérateurs temporels permettent de manipuler les domaines temporels (extension des relations de Allen) et permettent d'obtenir les données de l'entrepôt en fonction de valeurs temporelles (State).

Enfin, nous proposons des opérateurs spécifiques permettant de manipuler les ensembles d'états constituant les objets entrepôt (Current, Past, Archive, DisConnect, Connect). En outre, un opérateur de formatage (MakeSerie) permet des traitements et des transformations pour l'analyse directe des données de l'entrepôt (Agreg, ACum, AMove, ScaleUp, ScaleDown).

La contribution de cette algèbre réside principalement dans son expressivité facilitant l'interrogation l'entrepôt et permettant de transformer les évolutions sous une forme (série temporelle d'états) adaptée aux analyses.

Cependant, la prise de décision utilise des processus OLAP qui s'appuient souvent sur une organisation multidimensionnelle des données. Le chapitre V se propose de définir une démarche de transformation des objets de l'entrepôt en un schéma en étoile dans un magasin. Ces magasins de données multidimensionnelles servent de support aux analyses multidimensionnelles.

CHAPITRE V :
MODELISATION, ELABORATION ET MANIPULATION DES MAGASINS DE DONNEES

1 INTRODUCTION A L'ELABORATION DE MAGASINS DE DONNEES A PARTIR DE L'ENTREPOT

1.1 Objectif

Nous avons défini dans les chapitres II, III et IV un modèle spécifique de représentation des données de l'entrepôt, un mécanisme d'élaboration à partir d'une source globale et une algèbre pour la manipulation des données de l'entrepôt. Ce chapitre étudie la réorganisation constituant l'architecture des systèmes décisionnels que nous avons définie au chapitre I dans la section 5.5. En entrée, le module de réorganisation utilise l'information décisionnelle contenue dans l'entrepôt de données, afin de produire différents magasins de données en sortie.

Un magasin constitue une base spécialisée (il est orienté sujet [Codd 1993]) et supporte efficacement l'interrogation et l'analyse pour l'aide à la prise de décision. En particulier, un magasin de données doit être adapté aux processus OLAP couramment utilisés pour aider la prise de décision [Inmon 1994] [Kimball 1996]. Or ces processus s'appuient sur une représentation multidimensionnelle des données [Kimball 1996] [Agrawal, et al 1997] parce que cette organisation de l'information est bien adaptée à ce type d'activité : la modélisation multidimensionnelle offre une représentation des données conforme à la perception qu'en ont les analystes [Marcel 1998] et les systèmes qui en découlent ont des temps de réponses plus rapides [Kimball 1996].

Dans ce chapitre, nous focalisons notre étude sur ce type de magasins multidimensionnels. Notre objectif est de fournir les éléments nécessaires à la représentation, à la construction et à la manipulation d'un magasin multidimensionnel.

1.2 Existant

La majorité des travaux de recherche sur l'approche multidimensionnelle se situent dans un contexte ROLAP [Kimball 1996] [Agrawal, et al 1997], MOLAP [Gyssen, et al 1997] ou OOLAP [Buzydlowski et al. 1998] [Lehner 1998] [Pedersen, Jensen 1999]. Nous souhaitons définir un modèle conceptuel qui reporte ce choix au niveau logique.

Un magasin de données, organisé de manière multidimensionnelle, est élaboré à partir de l'entrepôt dont les données ne sont pas représentées suivant ce type d'organisation. La réorganisation multidimensionnelle des données constitue une tâche essentielle dans l'élaboration d'un système décisionnel [Golfarelli, et al 1998]. Cependant, peu de travaux proposent des solutions visant à cette réorganisation. Nous pouvons citer quelques travaux, constituant des premières tentatives dans ce sens.

- [Golfarelli, et al 1998] présente un mécanisme de transformation d'un modèle entité/relation en un schéma en étoile.
- [Tryfona, et al 1999] propose une modélisation conceptuelle des magasins de données multidimensionnelles. L'approche proposée consiste à utiliser le modèle "*starER*" basé sur

un schéma Entité/Association étendu aux concepts de faits et de dimensions inhérents aux schémas en étoile.

- [Bret, et al 2000] décrit une méthodologie pour modéliser multidimensionnellement les magasins de données orientées objet.

Ces travaux sont inadaptés à notre contexte. En effet, ils utilisent une source non temporelle (les données sont simplement datées par un attribut). Nous souhaitons proposer une solution qui tient compte des données temporelles et archivées de l'entrepôt. En outre, nous nous situons au niveau conceptuel pour nous démarquer des choix inhérents au niveau logique. [Tryfona, et al 1999] propose un modèle conceptuel intégrant de manière systématique la dimension temporelle ; notre modèle en constellation de fait est une généralisation des modèles en étoile tels que celui proposé dans [Tryfona, et al 1999]. Enfin, comme nous l'avons montré au chapitre I, les modèles multidimensionnels actuellement proposés n'intègrent pas l'ensemble des opérations multidimensionnelles.

1.3 Proposition

Dans ce chapitre, nous proposons un modèle conceptuel de représentation des magasins de données [Ravat, Teste, Zurfluh 2001], une démarche pour le passage de l'entrepôt orienté objet et temporel vers un magasin multidimensionnel et une algèbre pour la manipulation multidimensionnelle des magasins [Ravat, Teste, Zurfluh 2001].

Ce chapitre est organisé de la manière suivante.

- La section 2 décrit notre modèle conceptuel dédié aux magasins. Ce modèle permet une représentation multidimensionnelle de l'information.
- La section 3 présente l'élaboration d'un magasin. Cette élaboration est réalisée suivant une démarche pour transformer les données de l'entrepôt dans un magasin multidimensionnel.
- La section 4 fournit une algèbre permettant de manipuler un magasin de manière multidimensionnelle. Cette algèbre fédère l'ensemble des propositions faites jusqu'à ce jour, en supportant toutes les opérations multidimensionnelles proposées.

2 MODELISATION CONCEPTUELLE DES MAGASINS DE DONNEES

Le modèle multidimensionnel consiste à représenter les données dans un magasin sous la forme d'un schéma regroupant des faits et des dimensions.

2.1 Faits

Dans notre architecture des systèmes décisionnels, un magasin de données est spécialisé pour un type d'activité, un type d'analyse, un groupe d'utilisateurs. Il contient seulement les informations issues de l'entrepôt qui sont jugées pertinentes (par l'administrateur) pour répondre aux objectifs des utilisateurs.

Un magasin de données comporte donc un (ou plusieurs) **fait(s)**. Il s'agit du sujet de l'analyse, c'est à dire des différentes **mesures** de l'activité à analyser.

Définition :
Le schéma d'un **fait** F se définit par le couple (Nom^F, M^F) où

- Nom^F désigne le fait,
- $M^F=\{A^F_1, A^F_2,\ldots, A^F_f\}$ est un ensemble d'attributs représentant les mesures de l'activité contenues dans le magasin.

Remarque :
Les groupements de mesures dans les faits sont effectués en fonction de choix réalisés par l'administrateur ; les mesures sont groupées par thèmes.

> *EXEMPLE :* Nous souhaitons analyser des mouvements financiers dans les établissements hospitaliers. Les mesures de l'activité concernent
>
> - les subventions perçues par les établissements hospitaliers (montant des subventions perçues, montant des taxes payées sur ces subventions) et
> - le budget de ces établissements (budget global).
>
> L'administrateur définit deux faits :
>
> -F_1=(*Sub*, {*subventionG*, *taxe*}),
>
> -F_2=(*Bud*, {*budgetH*}).

2.2 Dimensions et hiérarchies

Les mesures de l'activité peuvent être observées selon différentes perspectives. Chaque perspective est modélisée par une **dimension** qui regroupe les **paramètres** des mesures de l'activité. Ces paramètres déterminent les niveaux de détail de l'analyse.

Les analyses multidimensionnelles utilisent des **hiérarchies de paramètres** pour visualiser les valeurs analysées à différents niveaux de détail (notamment lors des opérations de forage vers le haut ou vers le bas). Ces hiérarchies organisent les paramètres entre eux selon une dépendance de hiérarchie. Une dépendance de hiérarchie entre deux paramètres A_i et A_j indique qu'à une **position** (valeur) de A_i est associée exactement une position de A_j et à une position de A_j est associée une (ou plusieurs) position(s) de A_i.

Nous proposons de modéliser les dépendances de hiérarchie par des dépendances fonctionnelles ; il est important de ne pas confondre une dépendance de hiérarchie au sens de l'analyse multidimensionnelle avec les dépendances hiérarchiques définies dans les bases de données relationnelles [Delobel, Adiba 1982].

Définition :
Une **dépendance de hiérarchie** entre deux paramètres A_i et A_j, notée $A_i \Rightarrow A_j$, implique :

- une dépendance fonctionnelle entre A_i et A_j, notée $A_i \rightarrow A_j$,
- l'absence d'une dépendance fonctionnelle $A_j \nrightarrow A_i$.

> *EXEMPLE :* Soient deux paramètres "*region*" et "*departement*". Nous considérons les ensembles de positions admises pour chaque paramètre :

- Positions(*region*)={*Midi-Pyrénées*, *Aquitaine*, *Languedoc-Roussillon*}
- Positions(*departement*)={*Haute-Garonne*, *Tarn*, *Tarn-et-Garonne*, *Pyrénées-Orientales*, *Hérault*, *Gironde*, *Pyrénées-Atlantiques*, *Dordogne*}

On observe que *departement*→*region* et *region*↛*departement* (à une position de "*departement*" correspond exactement une position de "*region*" et à une position de "*region*" correspond plusieurs positions de "*departement*").

Il existe la dépendance de hiérarchie *departement*⇒*region*.

En se basant sur les dépendances de hiérarchie, il est possible de définir une hiérarchie.

<u>*Définition*</u> :
Une **hiérarchie** est définie sur un ensemble de paramètres d'une même dimension D par $H^D_i=<A^D_1, A^D_2,\ldots, A^D_r>$ où $A^D_1 \Rightarrow A^D_2 \Rightarrow \ldots \Rightarrow A^D_r$.

Un paramètre A^D_t est considéré comme **terminal** de H^D_i s'il n'existe pas un paramètre A^D_k de H^D_i tel que $A^D_k \Rightarrow A^D_t$.

<u>*EXEMPLE :*</u> Nous considérons l'ensemble des paramètres {*ville*, *departement*, *region*}. On peut définir les dépendances de hiérarchie suivantes : *ville*⇒ *departement* et *departement*⇒*region*

Il existe donc une hiérarchie sur cet ensemble de paramètres : H=<*ville*, *departement*, *region*>. Le paramètre "*ville*" est le paramètre terminal de la hiérarchie H.

Il est possible de définir plusieurs hiérarchies pour un même ensemble de paramètres ; on parle de **hiérarchies multiples**. Celles-ci organisent les paramètres en treillis.

Un magasin est donc constitué de dimensions dont les paramètres sont organisés selon une hiérarchie. Il s'agit des paramètres de l'analyse.

<u>*Définition*</u> :
Le schéma d'une **dimension** D se définit par le triplet (Nom^D, $Attribut^D$, H^D) où

- Nom^D désigne la dimension,
- $Attribut^D=\{A^D_1, A^D_2,\ldots, A^D_a\}$ est un ensemble d'attributs représentant les paramètres de l'activité contenues dans la dimension,
- $H^D=<H^D_1, H^D_2,\ldots, H^D_h>$ est la liste des hiérarchies définies sur la dimension, formant une hiérarchie multiple. H^D_1 est la **hiérarchie courante**, c'est à dire la hiérarchie utilisée pour visualiser les données de la dimension D.

L'ensemble H^D des hiérarchies associées à la dimension D est ordonnée. Ce classement permet de modéliser la hiérarchie courante H^D_1 utilisée par les utilisateurs pour visualiser la dimension (*cf.* section 4.1).

On pose $P^D=\bigcup_{k=1}^{h} attr(H^D_k)$ où $attr(H^D_k)$ est une fonction qui retourne l'ensemble des attributs d'une hiérarchie. Les attributs de la dimension n'appartenant pas aux hiérarchies de la dimension ($Attribut^D$-P^D) sont des **attributs faibles** ; ils n'interviennent pas pour définir un niveau de détail mais ils peuvent cependant être utilisés pour d'autres opérations telles que la sélection.

EXEMPLE : Le budget des hôpitaux peut être analysé selon la dimension géographique D_1 définie de la manière suivante :

Nom^{D1}="*Geo*",

$Attribut^{D1}$={*nomH*, *ville*, *departement*, *region*, *statut*},

H^{D1}=<$H^{D1}{}_1$> | $H^{D1}{}_1$=<*nomH*, *ville*, *departement*, *region*>

L'attribut "*statut*" est un attribut faible puisqu'il n'intervient pas dans la hiérarchie.

2.3 Schéma multidimensionnel

Chaque magasin de données est caractérisé par son schéma multidimensionnel. Nous définissons un **schéma multidimensionnel** comme un schéma formé de faits et de dimensions, parmi lesquels un fait et deux dimensions sont considérés courants. En effet, la visualisation de l'information contenue dans un schéma multidimensionnel est réalisée sous forme de tableaux ; nous présentons le principe de visualisation des données d'un schéma multidimensionnel dans la section 4.1.

Définition :
Un **schéma multidimensionnel** est défini par le n-uplet (Nom^S, *FAI*, *DIM*, *Param*) où

- Nom^S est le nom du schéma multidimensionnel,
- $FAI=<F_1, F_2,\ldots, F_p>$ est l'ensemble ordonné des faits ; F_1 est le **fait courant** dont les mesures peuvent être affichées aux utilisateurs,
- $DIM=\{D_1, D_2,\ldots, D_q\}$ est l'ensemble des dimensions,
- *Param* : $FAI \rightarrow 2^{DIM}$ est une fonction associant chaque fait à ses dimensions. Cette fonction renvoie la liste des dimensions associées à un fait, $\forall i \in [1..p]$, $Param(F_i)=<D_{i1}, D_{i2},\ldots, D_{iq}>$ avec $\forall j \in [i_1..i_q]$, $D_{ij} \in DIM$; $D_{1\,1}$ et $D_{1\,2}$ sont les **dimensions courantes** utilisées pour afficher les données du fait courant F_1.

Le schéma multidimensionnel comprend un ensemble ordonné de faits *FAI* afin de modéliser le fait courant F_1 dont les mesures sont visualisées par les utilisateurs. D'autre part, la fonction *Param* renvoie également un ensemble ordonné afin de modéliser les deux dimensions courantes D_{i1} et D_{i2} utilisées pour visualiser les données du schéma multidimensionnel.

Par conséquent, les concepts de faits courants et de dimensions courantes sont utiles lors de la visualisation des données du schéma. En effet, une constellation de faits rend plus complexe la visualisation des données par les utilisateurs. Nous étudions la visualisation des données et leur manipulation dans la section 4 de ce chapitre V.

EXEMPLE : Nous poursuivons l'exemple relatif aux analyses de mouvements financiers dans les établissements hospitaliers. Il est possible de définir le schéma multidimensionnel *Sh* suivant :

-*NomS*="*FinancesHopitaux*"

-$FAI=<F_1, F_2>$ où F_1=(*Sub*, {*subventionG*, *taxe*}),

F_2=(*Bud*, {*budgetH*})

-$DIM=\{D_1, D_2, D_3\}$ où D_1=(*Geo*, {*nomH*, ville, *departement*, *region*, *statut*}, H^{D1}),

D_2=(*Don*, {*nomOrg*, *type*}, H^{D2}),

D_3=(*Tps*, {*jour*, *mois*, *saison*, *trimestre*, *annee*}, H^{D3}).

-*Param* : *Param*(F_1)=<D_1, D_2, D_3>

Param(F_2)=<D_1, D_3>

Chaque dimension du schéma multidimensionnel est munie d'une hiérarchie, pouvant être multiple. Ces hiérarchies sont définies comme suit :

H^{D1}=<<*nomH*, *ville*, *departement*, *region*>>

H^{D2}=<<*nomOrg*, *type*>>

H^{D3}=<<*jour*, *mois*, *trimestre*, *annee*>>

Dans l'exemple de la section 4.1, nous décrivons la représentation tabulaire pour la visualisation des données de cet exemple de schéma multidimensionnel.

Nous complétons notre modélisation par un formalisme graphique. Cette représentation facilite la compréhension du schéma multidimensionnel en faisant apparaître visuellement la structure en constellation du schéma, ses sous-structures en étoiles et les hiérarchies des dimensions. Notre proposition consiste en trois catégories de graphes de représentation du modèle conceptuel :

- le **graphe en constellation** *GC*=(*NC*, *LC*) où
  - *NC* est un ensemble de nœuds représentant les faits et les dimensions du schéma en étoile,
  - *LC* est un ensemble de liens représentant les correspondances modélisées par la fonction *Param(F)* entre les faits et les dimensions.
- Les **graphes en étoile** (*p* graphes en étoile sont définis par schéma multidimensionnel) GE_k=(NE_k, LE_k) (avec $k \in [1..p]$), où
  - NE_k est un ensemble de nœuds représentant le fait F_k et les dimensions associée au fait par *Param*(F_k),
  - LE_k est l'ensemble des liens représentant les correspondances modélisées par la fonction *Param*(F_k) entre le fait F_k et les dimensions de *Param*(F_k).
- Les **graphes de hiérarchie** GH^{Di}_j=(NH^{Di}_j, LH^{Di}_j) avec $i \in [1..q]$ et $j \in [1..h]$ où
  - NH^{Di}_j est l'ensemble des paramètres présents dans la hiérarchie H^{Di}_j,
  - LH^{Di}_j est l'ensemble des liens représentant les dépendances de hiérarchie entre les paramètres de la hiérarchie.

EXEMPLE : Nous complétons l'exemple précédent en donnant les graphes associés au schéma multidimensionnel intitulé "*FinancesHopitaux*".

-La Figure 32 décrit le graphe en constellation issu du schéma multidimensionnel,

-La Figure 33 décrit les deux sous-graphes en étoile issu du schéma multidimensionnel,

-La Figure 34 décrit les trois graphes de hiérarchie représentant les dépendances de hiérarchie des paramètres contenus dans les trois dimensions D_1, D_2 et D_3.

Dans les graphes en constellation et en étoile, les nœuds représentant un fait sont distingués de ceux représentant une dimension par des symboles spécifiques : un fait est estampillé par une figure symbolisant un cube et une dimension par une figure symbolisant un repère à trois axes.

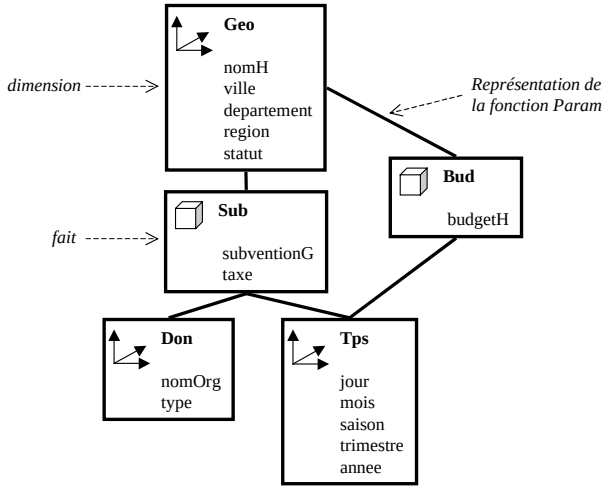


Figure 32 : Exemple de graphe en constellation (avec deux faits et trois dimensions).

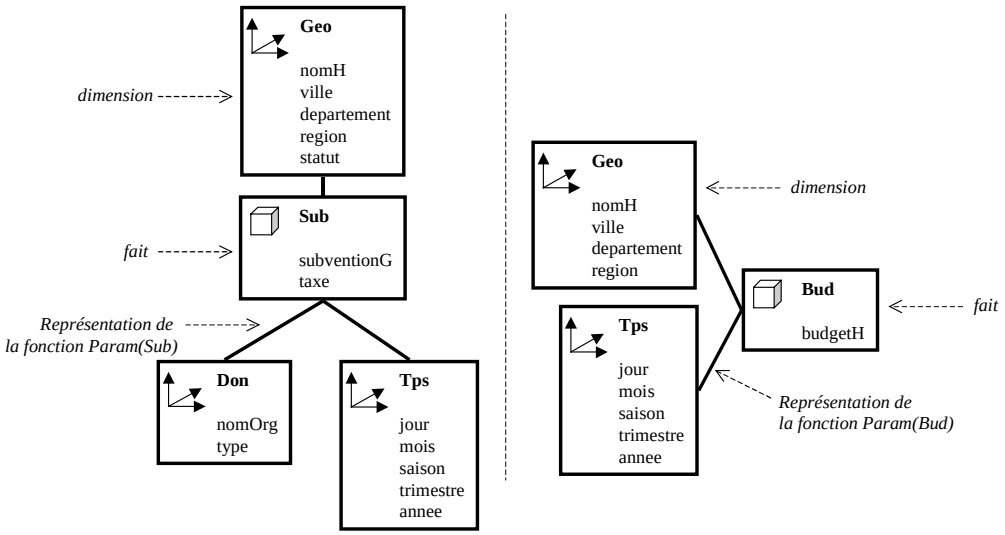


Figure 33 : Exemple de graphes en étoile.

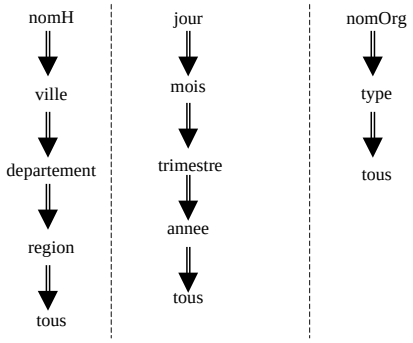


Figure 34 : Exemple de treillis des hiérarchies.

2.4 Synthèse

Nous avons défini un modèle conceptuel dédié aux magasins de données. Un magasin de données se caractérise par son schéma multidimensionnel. Il se compose :

- de faits (regroupant les mesures de l'activité analysée) et
- de dimensions (regroupant les paramètres de l'analyse) dont les paramètres sont organisés suivant des dépendances de hiérarchie.

Nous complétons notre modélisation par un formalisme graphique permettant de mieux appréhender l'information contenue dans le magasin.

La contribution de ce modèle est principalement la généralisation des différents modèles actuels en intégrant plusieurs faits (schéma en constellation) et en associant plusieurs hiérarchies aux dimensions (hiérarchies multiples). Ce modèle étendu supporte une algèbre de manipulation adaptée que nous décrivons à la section 4

3 DEMARCHE POUR LA TRANSFORMATION DES DONNEES DE L'ENTREPOT DANS UN MAGASIN

Selon notre architecture des systèmes décisionnels, les magasins de données sont élaborés à partir de l'entrepôt. Nous proposons de construire un magasin en suivant une démarche comportant quatre étapes :

- détermination des faits représentant les sujets analysés,
- détermination des dimensions représentant les perspectives de l'analyse,
- définition des granularités des données de l'analyse,
- organisation des paramètres des dimensions selon des dépendances de hiérarchie pour supporter les analyses à différents niveaux de détail.

3.1 Détermination des faits

La première étape consiste à déterminer le sujet de l'analyse, c'est à dire la liste *FAI* du schéma multidimensionnel qui caractérise le magasin. Le sujet est représenté par des faits qui sont issus des classes entrepôt. Autrement dit, l'élaboration des faits est réalisée par dérivations de classes entrepôt particulières dites **classes représentatives**. Une classe représentative est désignée par l'administrateur ; il s'agit d'une classe contenant l'information qui doit être analysée (mesures de l'activité).

L'entrepôt de données contient généralement plusieurs classes représentatives à partir desquelles sont créés les faits d'un magasin. On considère l'entrepôt de données (Nom^{ED}, C^{ED}, Env^{ED}, $Config^{ED}$).

Définition :
La **détermination d'un fait** F_i est réalisée par l'opération $Fact(nom^{Fi}, CR, Mes)$ où

- *nom* est le nom du fait,
- $CR \in C^{ED}$ est une classe représentative contenue dans l'entrepôt,
- $Mes=\{(A_i, f_i) \mid i \in [1..f]\}$ est un ensemble de couples (A_i, f_i) où A_i est une mesure de l'activité contenue dans le fait et f_i est la fonction qui calcule les valeurs de A_i en l'appliquant sur la classe représentative *CR*.

Les fonctions de calcul f_i permettent de calculer les valeurs des mesures contenues dans le fait. Ces valeurs sont obtenues à partir des objets d'une classe représentative *CR* en appliquant une fonction de calcul qui peut être :

- un attribut de *CR* (la mesure est calculée à partir de l'attribut lui-même),
- une opération de type fonction de *CR* (la mesure représente le résultat retourné par la fonction).

Ainsi, chaque mesure est construite à partir d'un attribut ou d'une opération de *CR*, ou bien à partir d'une fonction d'agrégation.

> *EXEMPLE :* Nous souhaitons élaborer le magasin de données caractérisé par le schéma multidimensionnel présenté à la section 2.3 et relatif au suivi des finances des établissements hospitaliers. Le magasin de données est construit à partir de l'entrepôt de données dont le schéma est décrit à la section 4.5 du chapitre II.
>
> L'administrateur souhaite construire un magasin contenant les mesures de l'activité suivante : le montant des subventions perçues, le montant des taxes payées sur ces subventions et le budget global des établissements. La classe représentative des faits est unique, il s'agit de la classe entrepôt "*Etablissement*". L'administrateur crée les deux faits suivants :
>
> $Fact(Sub, Etablissement, \{(subventionG, subvention.montant), (taxe, taxe)\}) \rightarrow F_1$
>
> $Fact(Bud, Etablissement, \{(budgetH, budget())\}) \rightarrow F_2$

Les valeurs contenues dans le fait sont calculées à partir des objets de la classe représentative. Cependant, en fonction de l'objectif décisionnel du magasin, il n'est pas systématiquement utile d'extraire dans un magasin l'ensemble des valeurs contenues dans l'extension de *CR*. Par conséquence, la détermination du fait peut être complétée par une sélection, qui indique le sous-ensemble des objets de la classe représentative qui doit participer à la génération des valeurs du magasin. L'opération de détermination du fait est alors définie par $Fact(nom^{Fi}, \sigma(CR, pred), Mes)$ où σ est l'expression d'une sélection portant sur *CR* et *pred* est un prédicat valide exprimant une condition.

> *EXEMPLE :* Nous reconsidérons l'exemple précédent. L'administrateur désire recopier dans le magasin uniquement les informations relatives aux établissements hospitaliers publics. Ainsi il redéfinit les faits de la manière suivante :
>
> $Fact(Sub, \sigma(Etablissement, statut=\text{"public"}),$
> $\{(subventionG, id(subvention.montant)), (taxe, id(taxe))\}) \rightarrow F_1$
>
> $Fact(Bud, \sigma(Etablissement, statut=\text{"public"}),$
> $\{(budgetH, budget())\}) \rightarrow F_2$

3.2 Détermination des dimensions

La seconde étape de notre démarche consiste à déterminer les différentes perspectives de l'analyse. Il s'agit de créer les dimensions. Autrement dit, cette étape consiste à définir l'ensemble *DIM* et la fonction *Param* qui associe chaque fait précédemment défini aux dimensions de l'ensemble *DIM*.

Les dimensions sont élaborées à partir de classes entrepôt dites **dépendantes** des classes représentatives des faits. Chaque classe représentative d'un fait possède un ensemble de classes dépendantes à partir desquelles il est possible de construire les dimensions du fait. Cet ensemble de classes dépendantes est construit automatiquement en suivant un principe de dépendance entre classes.

<u>*Définition*</u> :
Le **principe de dépendance** entre classes entrepôt permet de spécifier l'ensemble des classes dépendantes d'une classe représentative d'un fait. Une classe entrepôt $C_1 \in C^{ED}$ est dépendante d'une classe $C_2 \in C^{ED}$ (on note $C_1 \Rightarrow C_2$) si elle satisfait au moins une des règles suivantes :

- $C_1 = C_2$ (la classe représentative est elle-même une classe dépendante), (*R1*)
- C_1 est reliée à C_2 par une relation d'héritage telle que C_1 est sous-classe de C_2, (*R2*)
- C_1 est reliée à C_2 par une relation d'association de cardinalité $(x, 1)$, (*R3*)
- C_1 est reliée à C_2 par une relation de composition
  - C_1 est une classe composante de C_2 avec la cardinalité $(x, 1)$, (*R4'*)
  - C_1 est une classe composée de C_2 avec la cardinalité $(1, x)$. (*R4''*)

La contrainte sur les cardinalités des relations d'association et de composition permet de garantir l'unicité entre les valeurs d'une classe C_1 et les valeurs liées d'une classe C_2. Cette propriété est essentielle, car elle permet de relier dans le magasin les mesures issues de la classe représentative aux paramètres issus des classes dépendantes.

Le principe de dépendance respecte la propriété de transitivité. La **propriété de transitivité du principe de dépendance** permet de définir une classe C_1 dépendante d'une classe C_3 si la classe C_1 est dépendante d'une classe C_2 elle-même dépendante de C_3 ; soient $C_1 \in C^{ED}$, $C_2 \in C^{ED}$ et $C_3 \in C^{ED}$, si $C_1 \Rightarrow C_2$ et $C_2 \Rightarrow C_3$ alors $C_1 \Rightarrow C_3$.

A partir du principe de dépendance et de sa propriété de transitivité, on peut déterminer l'ensemble $Depend(CR)=\{CD_1, CD_2,\ldots, CD_m\}$ des classes dépendantes d'une classe représentative *CR*. Leur intérêt est de déterminer l'ensemble des classes entrepôt à partir desquelles peuvent être créées les dimensions.

> <u>*EXEMPLE :*</u> Nous considérons l'entrepôt de données décrit à la section 4.5 du chapitre II. Nous souhaitons identifier les classes dépendantes pour chaque classe de l'entrepôt. Pour chaque classe de cet entrepôt, l'ensemble des classes dépendantes peut être défini.
>
> $Depend(C^{Personne})=\{C^{Personne}, C^{Praticien}, C^{Cabinet}, C^{Service}, C^{Etablissement}\}$
>
> $Depend(C^{Praticien})=\{C^{Praticien}, C^{Cabinet}, C^{Service}, C^{Etablissement}\}$
>
> $Depend(C^{Cabinet})=\{C^{Cabinet}\}$
>
> $Depend(C^{Visite})=\{C^{Visite}, C^{Praticien}, C^{Cabinet}, C^{Service}, C^{Etablissement}\}$

Depend($C^{Service}$)={$C^{Service}$, $C^{Praticien}$, $C^{Cabinet}$ }

Depend($C^{Etablissement}$)={$C^{Etablissement}$}

Nous détaillons la détermination des classes dépendantes de *Personne*. Il est possible d'établir l'ensemble des classes dépendantes de *Personne* de la manière suivante :

- la classe *Personne* est dépendante d'elle même (*R1*),
- la classe *Praticien* qui hérite de *Personne* (*Praticien* est une sous-classe de *Personne*) est dépendante (*R2*),
- la classe *Cabinet*, reliée par une association de cardinalité (1,x) est dépendante de *Praticien*, qui par transitivité rend *Cabinet* dépendante de *Personne* (*R3*),
- de manière similaire, *Service* est caractérisée comme dépendante de *Personne*,
- la classe *Etablissement* est composée par la classe *Service* suivant une cardinalité (1, x), permettant d'affirmer que *Etablissement* est par transitivité dépendante de *Personne* (*R4"*).

Les dimensions associées au fait F_i obtenu suite à l'opération *Fact*(nom^{Fi}, *CR*, *DERIV*), sont créées à partir d'une classe dépendante $CD \in Depend(CR)$.

<u>*Définition*</u> :
La **détermination d'une dimension** D_i est réalisée par l'opération *Dim*(nom^{Di}, *CR*, *CD*, *Par*, *LF*) où

- *nom* est le nom de la dimension,
- *CR* est la classe représentative d'un fait,
- $CD \in Depend(CR)$ est une classe dépendante de *CR*,
- $Par=\{(A_i, f_i) \mid i \in [1..f]\}$ est un ensemble de couples (A_i, f_i) où A_i est attribut de la dimension et f_i est la fonction qui calcule les valeurs de A_i en l'appliquant sur la classe dépendante *CD*,
- $LF=\{F_1, F_2,\ldots\}$ est la liste des faits issus de *CR* et reliés par la fonction *Param* à la dimension D_i.

Les fonctions de calcul peuvent être :

- un attribut de *CD*,
- une opération de type fonction de *CD*.

<u>*EXEMPLE*</u> *:* Nous complétons l'exemple de la section 2.1 concernant un magasin comportant deux faits F_1 et F_2. La seconde étape permet de compléter le magasin par les dimensions qu'il contient. Chaque dimension d'un fait est élaborée à partir des classes dépendantes de la classe représentative *Etablissement* (puisque les deux faits du magasin ont été créés à partir de cette même classe). L'administrateur définit les dimensions suivantes :

Dim(*Geo*,
    Etablissement,
    Etablissement,
    { (*nomH*, *id*(*nom*)),
      (*ville*, *id*(*adresse.ville*)),

(*departement*, *depart*()),
(*region*, *region*()),
(*statut*, *id*(*statut*)) },
{F_1, F_2}) → D_1

Dim(*Don*,
Etablissement,
Etablissement,
{ (*nomOrg*, *id*(*subventions.institution*)),
(*type*, *id*(*subventions.type*)) },
{F_1}) → D_2

L'administrateur considère qu'il n'est pas pertinent (pour les utilisateurs du magasin) de lier la dimension D_2 relative aux organismes donateurs (organismes qui subventionnent les établissements hospitaliers) et le fait F_2 relatif au budget des établissements hospitaliers.

3.3 Cas de la dimension temporelle

Une contribution importante du modèle de l'entrepôt (que nous avons défini au chapitre II) est la prise en compte du temps au travers de mécanismes d'historisation et d'archivage des données. Ces données temporelles, essentielles lors des analyses et des processus OLAP [Inmon 1994] [Chaudhuri, Dayal 1997] sont exploitées au niveau des magasins de manière spécifique.

Dans notre entrepôt, le temps est modélisé de manière spécifique à partir d'un environnement et des filtres associés aux classes. Nous proposons un traitement semi-automatique permettant d'extraire les données temporelles afin de produire une dimension.

Un fait *F*, issu d'une classe représentative *CR*, peut être dimensionné par une dimension temporelle si et seulement si

- la classe représentative appartient à un environnement et
- les mesures du fait sont issues d'attributs appartenant aux filtres de la classe représentative.

Lorsque ces conditions sont vérifiées, l'entrepôt contient les évolutions des différents attributs participant à la création des mesures. L'administrateur peut créer une dimension temps.

<u>*Définition*</u> :
La **création d'une dimension temporelle** est réalisée par l'opération *DimT*(*nom*, *CR*, *ParT*, *LF*) où

- *nom* est le nom de la dimension temporelle,
- *CR* est la classe représentative d'un fait,
- $ParT=\{(A_i, U_i) \mid i\in[1..f]\}$ est un ensemble de couples (A_i, U_i) où A_i est attribut de la dimension et U_i est une unité temporelle utilisée au niveau des états des objets de la classe représentative du fait *F*,
- $LF=\{F_1, F_2,\ldots\}$ est la liste des faits, issus de *CR* et reliés par la fonction *Param* à la dimension temporelle.

<u>Pré-condition</u>
Cette opération est possible si et seulement si les conditions suivantes sont vérifiées :

- $\exists env\in Env^{ED} \mid CR\in C^{env}$,
- $\forall A_i\in M^F, \exists A_j\in Attribut^{CR} \mid origine(A_i)=A_j \wedge A_j\in Tempo^{CR}$ (où $origine(A_i)$ retourne l'attribut A_j à partir duquel est calculé la mesure A_i).

Pour construire la dimension temporelle, il convient de respecter la sémantique entre :

- les données temporelles modélisant des évolutions détaillées (états courants et passés) et
- les données temporelles modélisant des évolutions résumées (états archivés).

Pour cela, nous modélisons de manière spécifique la **dimension temporelle** ; elle se définit par $(Nom^D, Attribut^D, H^D)$ où

- Nom^D est le nom de la dimension temporelle,
- $Attribut^D=AttrRes\cup AttrDet$ avec
  - $AttrRes=\{A^D_1, A^D_2,\ldots, A^D_r\}$ est l'ensemble des attributs représentant les paramètres issus des attributs appartenant au filtre d'archives,
  - $AttrDet=\{A^D_{r+1}, A^D_{r+2},\ldots, A^D_a\}$ est l'ensemble des attributs représentant les paramètres issus des attributs n'appartenant pas au filtre d'archives,
- $H^D=<H^D_1, H^D_2,\ldots, H^D_h>$ est la liste des hiérarchies définies sur la dimension, formant une hiérarchie multiple.

La dimension temporelle ainsi définie comporte deux ensembles d'attributs ; les attributs de *AttrRes* constituent les paramètres relatifs aux évolutions archivées tandis que les attributs de *AttrDet* constituent les paramètres des évolutions détaillées. Au niveau logique (orienté objet), cette distinction donne lieu à la création de deux classes temporelles liées par une relation d'héritage comme l'illustre la Figure 35.

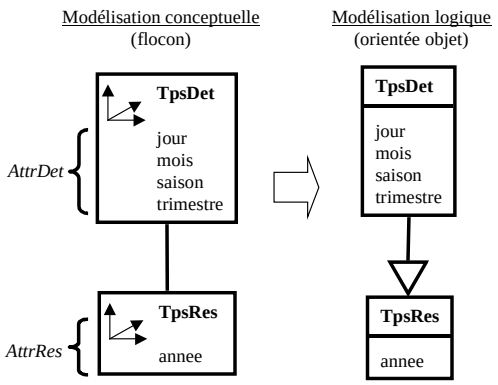


Figure 35 : Modélisations de la dimension temporelle.

Ainsi, les données issues des états courants et passés sont modélisées par la classe *TpsDet* te les données issues des états archivés sont modélisées par la classe *TpsRes*.

> EXEMPLE : Nous poursuivons l'exemple précédent. Les faits F_1 et F_2 sont issus de la classe représentative *Etablissement*. Or cette classe est contenue dans l'environnement *SuiviHopitaux* ; la classe *Etablissement* est munie d'un filtre temporel et d'un filtre d'archives conservant les évolutions des subventions et des budgets des hôpitaux. L'administrateur peut donc extraire une dimension temporelle :
>
> *DimT*(*Tps*,
> *Etablissement*,
> { (*jour*, *T_Unite*(*jour*)),
> (*mois*, *T_Unite*(*mois*)),
> (*saison*, *T_Unite*(*saison*)),
> (*trimestre*, *T_Unite*(*trimestre*)),
> (*annee*, *T_Unite*(*annee*))} },
> $\{F_1, F_2\}$) $\rightarrow D_3$
>
> La dimension temporelle D_3 contient les données courantes, passées et archivées extraites de *Etablissement* ; les attributs de la dimension sont définis de la manière suivante :
>
> $AttrRes^{D3}$={*annee*} et $AttrDet^{D3}$={*jour*, *mois*, *saison*, *trimestre*}

3.4 Définition des granularités

La troisième étape consiste à spécifier comment doivent être agrégées les données dans le magasin. L'énoncé précis des dimensions d'un fait détermine la granularité des mesures du fait ; par exemple, pour une dimension géographique, le choix du paramètre le plus fin (tel que "*departement*" ou "*ville*") détermine le niveau de détail avec lequel pourrons être observées les mesures de l'activité.

Suivant le niveau de granularité, plusieurs valeurs d'une mesure sont obtenues pour chaque combinaison des dimensions du fait. Ainsi, pour la dimension géographique, à la granularité "*departement*", il est possible d'obtenir plusieurs valeurs d'un budget (les valeurs des budgets des établissements du même département). Dans ce cas, il est donc impératif pour l'administrateur de spécifier le type regroupement devant être appliqué sur ces ensembles de valeurs.

Définition :
La **définition de la granularité** des mesures d'un fait est réalisée par l'opération $Gran(F_i, Agreg)$ où

- F_i est un fait,
- $Agreg=\{(A_i, f_i) \mid i\in[1..f]\}$ est un ensemble de couples (A_i, f_i) où A_i est une mesure de l'activité contenue dans le fait et f_i est une fonction d'agrégation telle que *avg(), sum(), count(), min(), max(), first(), last()* ou *nth(i)* qui permet de regrouper les valeurs.

EXEMPLE : Dans notre exemple, les dimensions D_1, D_2 et D_3 ne produisent qu'une valeur pour chaque combinaison ; l'agrégation n'est donc pas utilisée.

Cependant, si D_1 est redéfinie avec un niveau de détail moins fin :

Nom^{D1}="*Geo*",

$Attribut^{D1}$={*departement*, *region*, *statut*},

$H^{D1}=<H^{D1}_1> \mid H^{D1}_1=<departement, region>$

Dans ce cas, pour chaque position du paramètre *departement*, plusieurs valeurs sont obtenues pour les mesures. Il faut alors spécifier l'agrégation à appliquer sur le niveau de granularité le plus fin ; par exemple,

$Gran(F_1, \{(subventionG, avg), (taxe, sum)\})$ et $Gran(F_2, \{(budgetH, avg)\})$

3.5 Hiérarchisation des dimensions

La dernière étape dans l'élaboration d'un magasin de données consiste à organiser la hiérarchie des dimensions. Il s'agit de définir, pour chaque dimension, la liste H^D des hiérarchies des paramètres.

Définition :
La **hiérarchisation d'une dimension** est réalisée par l'opération $Hiera(D, H)$ où

- $D=(Nom^D, Attribut^D, H^D)$ est une dimension,
- $H=<A_1, A_2,\ldots, A_h>$ est une hiérarchie telle que $\forall i\in[1..h], A_i\in Attribut^D$.

Cette opération permet d'augmenter la dimension D par une nouvelle hiérarchie, $D=(Nom^D, Attribut^D, H^D\cup<H>)$.

EXEMPLE : Nous poursuivons l'exemple précédent. Le schéma multidimensionnel caractérisant le magasin est composé des faits F_1 et F_2 ainsi que des dimensions D_1, D_2 et D_3. Pour chaque dimension, l'administrateur construit une hiérarchie.

$Hiera(D_1, <nomH, ville, departement, region>) \rightarrow H^{D1}_1$

$Hiera(D_2, <nomOrg, type>) \rightarrow H^{D2}_1$

$Hiera(D_3, <jour, mois, trimestre, annee>) \rightarrow H^{D3}_1$

Cette opération nécessite une expertise de l'administrateur qui doit analyser les dépendances fonctionnelles entre les paramètres des dimensions. Il s'agit d'une tâche importante car elle détermine les différents niveaux de détail auxquels pourront être observés les mesures des faits lors des analyses.

3.6 Synthèse

Nous avons proposé une démarche d'élaboration d'un magasin de données caractérisé par son schéma multidimensionnel :

- déterminer les faits (chaque fait est constitué de mesures obtenues par extraction sur une classe entrepôt dite représentative du sujet),
- déterminer les dimensions (chaque dimension est obtenue à partir des classes dépendantes d'une classe représentative d'un fait et la dimension créée est associée aux faits précédemment définis),
- définir les granularités des mesures des faits en fonction des paramètres de plus bas niveau des dimensions associées,
- hiérarchiser les dimensions (les attributs des dimensions sont organisés en hiérarchies suivant les dépendances de hiérarchie).

La proposition présentée dans cette section intègre les particularités de notre modèle de données pour l'entrepôt. Une dimension temporelle peut être définie lorsque la classe représentative d'un fait appartient à un environnement. Cette dimension temporelle contient les données issues des états courants, passés et archivés des objets entrepôt de la classe représentative.

4 MANIPULATION DES DONNEES D'UN MAGASIN MULTIDIMENSIONNEL

Dans l'architecture des systèmes décisionnels, les magasins de données (dédiés à un groupe d'analystes) servent de base aux analyses multidimensionnelles. Pour cela, nous avons défini un modèle multidimensionnel généralisé qui représente l'information en constellation de faits et de dimensions munies de hiérarchies multiples (contrairement aux approches habituelles qui utilisent de simples schémas en étoile comportant un seul fait et souvent une seule hiérarchie par dimension).

Ce modèle supporte une algèbre de manipulation des données multidimensionnelles. Les approches actuelles [Kimball 1996] [Agrawal, et al 1997] [Gyssen, et al 1997] [Buzydlowski et al. 1998] [Pedersen, Jensen 1999] proposent différentes opérations de manipulation suivant les concepts du modèle (relation, objet, hypercube…). Nous proposons dans cette section d'unifier ces opérations en définissant un ensemble d'opérateurs permettant de réaliser les principales manipulations utilisées dans les processus OLAP. En outre, nous adaptons ces opérations à notre concept de schéma multidimensionnel qui organise les magasins sous la forme d'une constellation de faits.

4.1 Représentation tabulaire du schéma multidimensionnel

A notre sens, la représentation sous forme de tableau est la vision la plus simple et la plus intuitive à laquelle les utilisateurs d'un magasin de données sont habitués ; il s'agit de la représentation des données habituellement adoptée [Agrawal, et al 1997], [Gyssen, et al 1997]

[Lehner 1998]. Nous choisissons également de visualiser les données d'un schéma multidimensionnel (qui caractérise un magasin) en deux dimensions dans un tableau.

Soit le schéma multidimensionnel $Sh=(Nom^S, FAI, DIM, Param)$ où

- Nom^S est le nom du schéma multidimensionnel et
- $FAI=<F_1, F_2,\ldots, F_p>$ est l'ensemble des faits qui composent le schéma,
- $DIM=\{D_1, D_2,\ldots, D_q\}$ est l'ensemble des dimensions du schéma,
- *Param* est la fonction qui permet de relier chaque fait à ses dimensions telle que $\forall i\in[1..p]$, $Param(F_i)=<D_{i1}, D_{i2},\ldots , D_{iq}>$ avec $\forall j\in[i_1.. i_q]$, $D_{ij}\in DIM$,

F_1 est le fait courant dont les mesures sont visualisées par les utilisateurs. $D^{F1}{}_1$ et $D^{F1}{}_2$ sont les deux dimensions courantes dont les paramètres sont utilisés pour visualiser les mesures du fait courant suivant leur hiérarchie courante.

La Figure 36 illustre l'affichage initial du schéma multidimensionnel *Sh*. Le schéma multidimensionnel est organisé en colonnes, lignes et plans.

- Le fait courant est affiché. La liste des autres faits est disponible sur la droite de la figure. On peut permuter les faits pour changer les données présentées.
- Les deux dimensions courantes sont affichées en ligne et colonne, définissant ainsi un plan. La liste des autres dimensions est disponible en bas de la figure. On peut ensuite permuter les dimensions pour changer de plan.
- Pour les deux dimensions courantes, les paramètres sont affichés suivant la hiérarchie courante (initialement, seul le paramètre représentant le niveau de détail le moins fin de la hiérarchie courante est affiché).
- Les mesures du fait courant sont placées à l'intersection d'une ligne et d'une colonne.

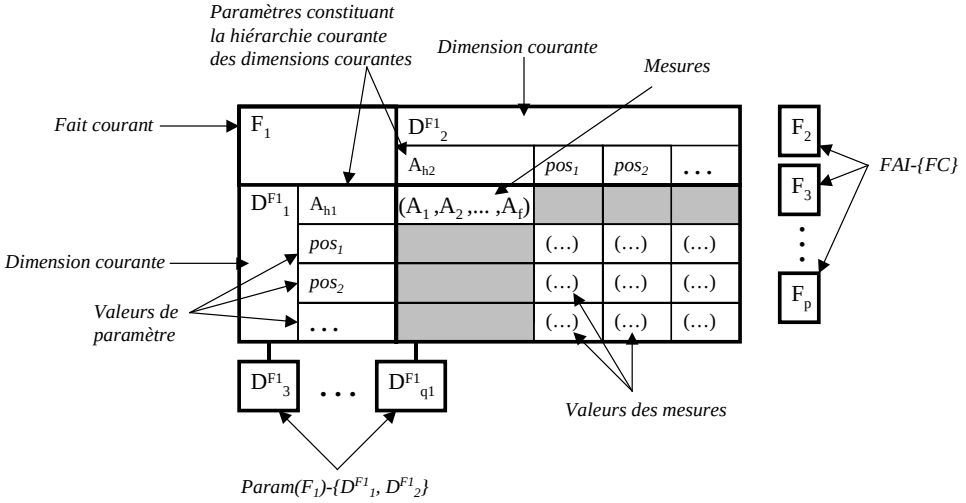


Figure 36 : Représentation tabulaire d'un schéma multidimensionnel.

> <u>EXEMPLE</u> *:* Nous reprenons le magasin de données caractérisé par le schéma multidimensionnel présenté à la section 2.3 et relatif au suivi des finances des établissements hospitaliers. Le tableau suivant présente les données du fait *Sub* contenu dans le schéma.
>
> Pour le paramètre *region*, nous notons *MP* la position (valeur) *Midi-Pyrénées*, *Aq* la position *Aquitaine* et *LR* la position *Languedoc-Roussillon*.

Sub		Don				Bud
		type	*public*	*privée*	*association*	
Geo	**region**	(subventionG, taxe)				
	MP		(250,25)	(200,70)	(150,15)	
	Aq		(300,40)	(280,90)	(70,10)	
	LR		(200,30)	(210,80)	(50,5)	
Tps.annee = 2000						

Tableau 20 : Représentation des subventions perçues par les établissements hospitaliers, pour l'année 2000, par région et par type d'organisme donateur.

Dans la suite de ce développement, nous montrons les différents types de manipulations applicables sur les schémas multidimensionnels. Nous adaptons les principaux opérateurs proposés dans la littérature aux schémas multidimensionnels tels que nous les définissons (constellation de faits) et nous introduisons d'autres opérateurs spécifique à notre type de modélisation. Par conséquent, nous proposons :

- des opérations classiques dans les bases de données issues de l'algèbre relationnelle,
- des opérations permettant de modifier la granularité des données affichées, c'est à dire de transformer les données pour les observées suivants différents niveaux de détail,
- des opérations permettant de modifier les structures affichées, c'est à dire de transformer les faits, les dimensions et les hiérarchies pour visualiser les données suivants différentes perspectives.

4.2 Adaptation des opérations classiques

Il est possible de **renommer** les noms d'attributs, c'est à dire les noms des mesures et des paramètres. Il est également possible de renommer les dimensions et les faits. D'autre part, les **suppressions** d'attributs (c'est à dire de mesures ou de paramètres), de faits ou de dimensions sont supportées par notre algèbre.

L'**ajout d'une mesure** dans un fait est réalisable. Il s'agit de mesures calculées dynamiquement (par l'intermédiaire d'une expression de calcul qui combine d'autres mesures) sur le magasin. Par exemple, si l'on dispose d'une mesure représentant le montant d'une vente et d'une autre mesure représentant la quantité vendue, il est alors possible d'ajouter une mesure, calculée dynamiquement (*vente * quantité*) et représentant le montant total de la vente.

Nous adaptons au schéma multidimensionnel les opérations des langages classiques en base de données [Chrisment, et al 1999] (plus précisément, il s'agit d'opérations issues de l'algèbre relationnelle). L'intérêt de ces opérations (sélection, jointure,…) est d'offrir aux utilisateurs (décideurs) des mécanismes flexibles de manipulation des informations nécessaires ; ces opérations permettent de manipuler les données comme dans une base de données traditionnelle.

Les opérations ensemblistes d'**union**, d'**intersection** et de **différence** sont utilisables avec des schémas multidimensionnels (on impose l'égalité des deux schémas). Elles confèrent à notre algèbre la puissance de manipulation de l'algèbre relationnelle en offrant la possibilité de combiner différents schémas multidimensionnels.

L'opération de **sélection** (*Slice and Dice* qui signifie littéralement "*couper des tranches*"), permet de sélectionner un sous-schéma. Elle correspond à l'opération de sélection de l'algèbre relationnelle mais s'applique ici à un schéma multidimensionnel.

Les opérateurs **produit cartésien** et la **jointure** peuvent s'appliquer aux schémas multidimensionnels. Ils sont utiles pour associer des données provenant de schémas distincts.

EXEMPLE : Considérons le schéma multidimensionnel *Sh* décrit au Tableau 20 ainsi que le schéma *Sh'* représentant la quantité de population par région et par année. Ce dernier est décrit de la manière suivante :

- $FAI=<F_1>$ avec $F_1=(Pop, \{nbHab\})$
- $DIM=\{D_1, D_2\}$ avec $D_1=(Geo, \{nomH, ville, departement, region, statut\}, <<nomH, ville, departement, region>>)$ et $D_2=(Tps, \{jour, mois, saison, trimestre, annee\}, <<jour, mois, trimestre, annee>, <jour, saison>>)$
- $Param(F_1)=<D_1, D_2>$

Un utilisateur qui souhaite réaliser des analyses concernant les subventions des hôpitaux tout en observant les quantités de population associées au positionnement géographique des établissements peut réaliser la jointure entre *Sh* et *Sh'* exprimée par l'expression suivante :

Join(*Sh*, *Sh'*, *Sh.Geo.region=Sh'.Geo.region* $\wedge$ *Sh.Tps.annee=Sh'.Tps.annee*)

Le résultat obtenu est décrit par le Tableau 21.

Sub		**Don**			
		type	*public*	*privée*	*association*
Geo	**region**	(subventionG, taxe, nbHab)			
	MP		(250,25, 2800000)	(200,70, 2800000)	(150,15, 2800000)
	Aq		(300,40, 3000000)	(280,90, 3000000)	(70,10, 3000000)
	LR		(200,30, 2300000)	(210,80, 2300000)	(50,5, 2300000)
Tps.annee = 2000					

Tableau 21 : Résultat de l'opération de jointure.

4.3 Transformation de la granularité des données

Ces opérations permettent de visualiser les mesures d'un schéma multidimensionnel en modifiant leur niveau de détail (granularité). Elles consistent à augmenter ou à diminuer les paramètres de l'analyse conformément aux hiérarchies des dimensions.

4.3.1 Opérations de forage

Les opérations de forage sont parmi les plus importantes du processus OLAP. Elles permettent d'augmenter ou de diminuer ("*forer*") le détail avec lequel l'information est représentée ; ces opérations de forage manipulent les différents niveaux de granularité des données multidimensionnelles. Plus précisément, le forage consiste à ajouter ou supprimer des paramètres d'une hiérarchie associée à une dimension.

L'opération de forage vers le bas consiste à introduire un paramètre de granularité inférieure dans la hiérarchie courante d'une dimension. L'utilisateur commence par analyser les données à un faible niveau de détail, puis l'utilisateur analyse à un niveau de détail inférieur.

Définition :
Le **forage vers le bas** introduit un niveau de détail inférieur dans la hiérarchie courante d'une dimension. La syntaxe de l'opérateur est la suivante :

DrillDown(Sh, D_i, A_j) = Sh'

Entrée
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$D_i \in DIM$ est une dimension du schéma multidimensionnel,

$A_j \in Attribut^{Di}$ est un paramètre de la dimension D_i.

Sortie
Sh'=(Nom^S, FAI, DIM', $Param$) avec DIM' issue de DIM et dont la dimension D_i est munie d'une nouvelle hiérarchie courante telle que H_{i1}=<A_j>$\cup H_{i1}$. Le paramètre A_j est ajouté à la hiérarchie courante.

EXEMPLE : Nous considérons le schéma multidimensionnel décrit au Tableau 24. L'utilisateur souhaite visualiser les données en descendant d'un niveau de détail ; il souhaite afficher les données suivant le paramètre *trimestre* en plus de celui de *annee* sur la dimension *Tps*. L'utilisateur définit l'opération suivante :

DrillDown(*Sh*, *Tps*, *trimestre*)

Le résultat de cette opération est décrit dans le tableau suivant.

Sub			**Tps**											
		annee	*1998*				*1999*				*2000*			
		trimestre	*T1*	*T2*	*T3*	*T4*	*T1*	*T2*	*T3*	*T4*	*T1*	*T2*	*T3*	*T4*
Geo	**region**	(subventionG, taxe)												
	MP		(50, 5)	(50, 4)	(50, 5)	(50, 5)	(60, 6)	(70, 7)	(50, 6)	(60, 6)	(60, 6)	(70, 7)	(50, 6)	(70, 6)
	Aq		(80, 10)	(60, 7)	(40, 8)	(100, 10)	(90, 10)	(80, 10)	(60, 10)	(80, 10)	(70, 10)	(70, 10)	(50, 10)	(110, 10)
	LR		(50, 8)	(40, 5)	(40, 5)	(40, 5)	(60, 9)	(50, 6)	(20, 4)	(50, 6)	(60, 10)	(50, 8)	(40, 4)	(50, 8)
Don.Type = "public"														

Tableau 22 : Résultat du forage vers le bas au paramètre *trimestre* sur la dimension *Tps*.

Le forage vers le haut permet de représenter les mesures à une granularité moins fine. L'opération ajoute un paramètre hiérarchiquement supérieur à la hiérarchie courante d'une dimension.

Définition :
Le **forage vers le haut** introduit un niveau de détail supérieur dans la hiérarchie d'une dimension. La syntaxe de l'opérateur est la suivante :

RollUp(Sh, D_i, A_j) = Sh'

Entrée
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$D_i \in DIM$ est une dimension du schéma multidimensionnel,

$A_j \in Attribut^{Di}$ est un paramèttre de la dimension D_i.

Sortie
Sh'=(Nom^S, FAI, DIM', $Param$) avec DIM' issue de DIM et dont la dimension D_i est munie d'une nouvelle hiérarchie courante telle que $H_{i1}=H_{i1} \cup <A_j>$. Le paramètre A_j est ajouté à la hiérarchie courante.

EXEMPLE : Nous poursuivons l'exemple précédent. Le Tableau 22 décrit le schéma *Sh* obtenu après un forage vers le bas. L'utilisateur réalise maintenant un forage vers le haut pour revenir à un niveau de granularité moins fin.

RollUp(*Sh*, *Tps*, *annee*)

Le Tableau 24 décrit le schéma obtenu.

4.3.2 Opération d'agrégation

L'opération d'agrégation permet de regrouper les mesures d'un fait suivant des valeurs de paramètres. Une fonction d'agrégation telle que *sum*, *avg* ou *count* peut ensuite s'appliquer sur les mesures regroupées.

Définition :
L'**agrégation** consiste à regrouper les valeurs prises par les mesures d'un fait suivant des valeurs de paramètres. La syntaxe de l'opérateur est la suivante :

Aggregate(Sh, F_i, $Func$, $Group$) = Sh'

Entrées
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$F_i \in FAI$ est un fait du schéma Sh,

$Func$=<(A_1, f_1), (A_2, f_2),…, (A_f, f_f)> est la liste des fonctions appliquées sur les groupes de valeurs des mesures du fait F_i, $\forall k \in [1..f]$, $A_k \in M^{Fi}$.

$Group$={P_1, P_2,…, P_g} est l'ensemble des paramètres utilisés pour regrouper les mesures.

Sortie
Sh'=(Nom^S, FAI', DIM', $Param'$) avec FAI'=<F_i>, DIM'={D_j | $D_j \in Param(F_i)$} et $Param'(F_i)=Param(F_i)$. Le résultat est un schéma multidimensionnel, réduit au sous-schéma en étoile comportant le seul fait F_i et les dimensions associées au fait. Les valeurs des mesures sont obtenues par agrégation des valeurs regroupées des mesures du fait F_i, en appliquant les fonctions d'agrégation f_1, f_2,…, f_f.

Les données de granularité minimale peuvent être **pré-agrégées** suivant les différentes combinaisons d'attributs appartenant aux dimensions du schéma multidimensionnel. Les pré-agrégats sont utilisés pour accélérer le traitement des opérations d'agrégation car il est inutile de revenir aux données les plus détaillées en utilisant des données d'un niveau intermédiaire. Cependant, ces pré-agrégats présentes des inconvénients : ils occupent un espace de stockage important (ce qui limite leur nombre) et il est difficile de prévoir les pré-agrégats à stocker (avec la fonction d'agrégation adaptée) [Harinarayan, et al 1996].

4.3.3 Opération de calcul des totaux

Cette opération permet de calculer des sous-totaux et un total final pour les mesures d'un fait selon une combinaison de deux dimensions.

Définition :
Le **calcul des totaux** d'un fait permet d'afficher les sous-totaux et le total final des mesures du fait courant, pour les dimensions courantes. La syntaxe de l'opérateur est la suivante :

Cube(*Sh*) = *Sh'*

Entrées
Sh=(Nom^S, *FAI*, *DIM*, *Param*) est un schéma multidimensionnel.

Sortie
Sh' est le schéma multidimensionnel résultat auquel les sous-totaux et le total final des mesures du fait courant *FC* sont ajoutés pour les dimensions courantes.

EXEMPLE : Nous considérons le schéma multidimensionnel décrit au Tableau 24. L'utilisateur souhaite visualiser les subventions des organismes publics en affichant les totaux de chaque ligne et chaque colonne. Il réalise donc l'opération suivante :

Cube(*Sh*) = *Sh'*

Le tableau suivant illustre le résultat obtenu. Une ligne et une colonne supplémentaires sont ajoutées pour afficher les sous-totaux ; elles représentent l'ensemble des positions pour les années (colonnes) et les régions (lignes). La position *tous*, ajoutée dans le tableau résultat, représente l'ensemble des positions du paramètre.

Sub		**Tps**				
		annee	*1998*	*1999*	*2000*	*tous*
Geo	**region**	(subventionG, taxe)				
	MP		(200,19)	(240,25)	(250,25)	(690,69)
	Aq		(280,35)	(310,40)	(300,40)	(890,115)
	LR		(170,23)	(180,25)	(200,30)	(550,78)
	tous		(650,77)	(730,90)	(750,95)	(2130,262)
Don.Type = "public"						

Tableau 23 : Résultat des calculs des totaux.

4.4 Transformation de la structure des données

Les opérations de transformation permettent de changer la représentation du schéma multidimensionnel en modifiant sa structure. L'objectif de ces opérations est de manipuler le schéma afin de mieux appréhender les informations.

4.4.1 Opérations de rotation

Cette opération est classique en analyse multidimensionnelle et permet d'analyser les données selon différentes perspectives. Cette opération s'applique aux dimensions d'un schéma multidimensionnel.

Définition :
La **rotation de dimensions** inverse deux dimensions dans la fonction *Param*(*F*). La syntaxe de l'opérateur est la suivante :

DRotate(Sh, F, D_i, D_j) = Sh'

Entrées
$Sh=(Nom^S, FAI, DIM, Param)$ est un schéma multidimensionnel,

$F \in FAI$ est un fait du schéma multidimensionnel,

$D_i \in Param(F)$ et $D_j \in Param(F)$ sont deux dimensions telles que $Param(F)=<\ldots, D_i, \ldots, D_j, \ldots>$.

Sortie
$Sh'=(Nom^S, FAI, DIM, Param')$ est le schéma multidimensionnel résultat tel que $Param'(F)=<\ldots, D_j, \ldots, D_i, \ldots>$.

EXEMPLE : Nous considérons l'exemple de schéma multidimensionnel décrit au Tableau 20. L'utilisateur souhaite visualiser les subventions des organismes publics suivant les deux dimensions du lieu géographique et du temps. Il définit l'opération de rotation des dimensions *Don* et *Tps* :

DRotate(*FinancesHopitaux*, *Sub*, *Don*, *Tps*)=*Sh'*

Dans le schéma *Sh'* résultant de cette rotation, les deux dimensions *Don* et *Tps* sont permutées de telle sorte que *Tps* devienne une dimension courante ; autrement dit, les valeurs des paramètres de *Geo* et *Tps* sont visibles. Le tableau suivant illustre le résultat obtenu.

Sub		**Tps**			
		annee	*1998*	*1999*	*2000*
Geo	**region**	(subventionG, taxe)			
	MP		(200,19)	(240,25)	(250,25)
	Aq		(280,35)	(310,40)	(300,40)
	LR		(170,23)	(180,25)	(200,30)
Don.Type = "public"					

Tableau 24 : Résultat de la rotation des dimensions *Don* et *Tps*.

Notre modélisation des magasins adopte des schémas multidimensionnels en constellation de faits ; autrement dit, plusieurs faits peuvent constituer un même schéma. Par conséquent, nous

étendons cet opérateur pour l'appliquer sur les faits de manière à effectuer des permutations sur le schéma multidimensionnel qui comporte plusieurs faits.

Définition :
La **rotation de faits** inverse deux faits dans *FAI*. La syntaxe de l'opérateur est la suivante :

FRotate(Sh, F_i, F_j) = Sh'

Entrées
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$F_i \in FAI$ et $F_j \in FAI$ sont deux faits Sh tels que FAI=<…, F_i,…, F_j,…>.

Sortie
Sh'=(Nom^S, FAI', DIM, $Param$) est le schéma multidimensionnel résultat dans lequel FAI'=<…, F_j,…, F_i,…>.

En outre, notre modèle de données pour les magasins associe plusieurs hiérarchies aux dimensions. Par conséquent, nous définissons une opération de rotation sur les hiérarchies des dimensions.

Définition :
La **rotation de hiérarchies** inverse deux hiérarchies d'une dimension. La syntaxe de l'opérateur est la suivante :

HRotate(Sh, D, H^D_i, H^D_j) = Sh'

Entrées
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$D \in DIM$ est une dimension du schéma multidimensionnel,

$H^D_i \in H^D$ et $H^D_j \in H^D$ sont deux hiérarchies associées à D telles que H^D=<…, H^D_i,…, H^D_j,…>.

Sortie
Sh'=(Nom^S, FAI, DIM', $Param$) est le schéma multidimensionnel résultat dans lequel la dimension D' issue de D est définie par D'=(Nom^D, $Attribut^D$, <…, H^D_j,…, H^D_i,…>).

4.4.2 Opération de permutation

L'opération de permutation s'applique sur les positions (valeurs) des paramètres des dimensions. Elle permet de changer l'ordre des positions affichées. L'intérêt de cette opération est qu'elle sert à réorganiser les données contenues dans les dimensions afin de mettre en évidence certaines valeurs de mesures ou de paramètres.

On introduit une fonction $Pos(A_i)$=<pos_1, pos_2,…, pos_s> ; cette fonction retourne l'ensemble ordonné des positions d'un paramètre A_i.

Définition :
La **permutation** inverse deux positions d'un paramètre. La syntaxe de l'opérateur est la suivante :

Switch(Sh, D, A_i, pos_{j1}, pos_{j2}) = Sh'

Entrées
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$D \in DIM$ est une dimension du schéma multidimensionnel,

$A_i \in Attribut^D$ est un paramètre de la dimension D,

pos_{j1} et pos_{j2} sont deux positions du paramètre A_i telles que $Pos(A_i)$=<…, pos_{j1},…, pos_{j2},…>.

Sortie
Sh'=(Nom^S, FAI, DIM, $Param$) est le schéma multidimensionnel dans lequel les positions pos_{j1} et pos_{j2} du paramètre A_i ont été inversées $Pos(A_i)$=<…, pos_{j2},…, pos_{j1},…>.

EXEMPLE : Nous poursuivons l'exemple du schéma multidimensionnel décrit par le Tableau 24. L'utilisateur souhaite visualiser les subventions des organismes publics en classant les régions par ordre alphabétique. Il réalise donc les permutations suivantes :

Switch(*Sh*, *Geo*, *region*, *MP*, *Aq*) = *Sh'*

Switch(*Sh'*, *Geo*, *region*, *MP*, *LR*) = *Sh''*

Le tableau suivant illustre le résultat obtenu.

Sub		**Tps**			
		annee	*1998*	*1999*	*2000*
Geo	**region**	(subventionG, taxe)			
	Aq		(280,35)	(310,40)	(300,40)
	LR		(170,23)	(180,25)	(200,30)
	MP		(200,19)	(240,25)	(250,25)
Don.Type = "public"					

Tableau 25 : Résultat des permutations des positions de l'attribut *region*.

4.4.3 Opération d'emboîtement

L'emboîtement permet de modifier l'ordre des paramètres d'une dimension. Il s'agit de permuter deux paramètres d'une dimension. L'intérêt de cette opération réside dans la possibilité pour l'utilisateur de modifier l'ordre d'importance entre les paramètres en modifiant l'ordre hiérarchique des paramètres.

Définition :
L'**emboîtement** permute deux paramètres dans la hiérarchie H^D associée à une dimension D. La syntaxe de l'opérateur est la suivante :

Nest(Sh, D, A_i, A_j) = Sh'

Entrées
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$D \in DIM$ est une dimension du schéma multidimensionnel,

$A_i \in Attribut^D$ et $A_j \in Attribut^D$ sont deux paramètres de la dimension D tels que $H^D{}_1$=<…, A_i,…, A_j,…>.

Sortie
Sh'=(Nom^S, FAI, DIM', $Param$) est le schéma multidimensionnel résultat dans lequel la dimension D' issue de D est telle que D'=(Nom^D, $Attribut^D$, H^D) où $H^D{}_1$=<…, A_j,…, A_i,…>.

EXEMPLE : Nous considérons le schéma multidimensionnel décrit par le Tableau 22. L'utilisateur souhaite visualiser les subventions des organismes publics par *trimestre*, tout en conservant le paramètre *annee*. Il réalise donc l'emboîtement suivant :

Nest(*Sh*, *Tps*, *annee*, *trimestre*) = *Sh'*

Le tableau suivant illustre le résultat obtenu.

Sub		**Tps**												
		trimestre	*T1*			*T2*			*T3*			*T4*		
		annee	*1998*	*1999*	*2000*	*1998*	*1999*	*2000*	*1998*	*1999*	*2000*	*1998*	*1999*	*2000*
Geo	**region**	(subventionG, taxe)												
	MP		(50, 5)	(60, 6)	(60, 6)	(50, 4)	(70, 7)	(70, 7)	(50, 5)	(50, 6)	(50, 6)	(50, 5)	(60, 6)	(70, 6)
	Aq		(80, 10)	(90, 10)	(70, 10)	(60, 7)	(80, 10)	(70, 10)	(40, 8)	(60, 10)	(50, 10)	(100, 10)	(80, 10)	(110, 10)
	LR		(50, 8)	(60, 9)	(60, 10)	(40, 5)	(50, 6)	(50, 8)	(40, 5)	(20, 4)	(40, 4)	(40, 5)	(50, 6)	(50, 8)
Don.Type = "public"														

Tableau 26 : Résultat de l'emboîtement du paramètre *trimestre* dans le paramètre *annee*.

4.4.4 Opérations d'enfoncement et de retrait

L'enfoncement consiste à transformer un paramètre d'une dimension en mesure d'un fait et le retrait réalise l'opération inverse. Ces opérations permettent de traiter de manière totalement symétrique les mesures et les paramètres d'une analyse. L'utilisateur est ainsi plus libre d'orienter l'analyse vers ce qui lui semble le plus pertinent.

Définition :
L'**enfoncement** transforme un paramètre d'une dimension en une mesure dans un fait. La syntaxe de l'opérateur est la suivante :

Push(Sh, D_i, A_j, F_k) = Sh'

Entrées
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$D_i \in DIM$ est une dimension du schéma multidimensionnel,

$A_j \in Attribut^{Di}$ est un attribut de la dimension D_i,

$F_k \in FAI$ est un fait du schéma multidimensionnel.

Sortie
Sh'=(Nom^S, FAI', DIM', $Param$) est le schéma multidimensionnel obtenu dans lequel la dimension D_i' issue de D_i est définie par D_i'=(Nom^{Di}, $Attribut^{Di}$-{A_j}, {H' | $\exists H \in H^D$, H'=H-<A_j>}) et A_j est une mesure du fait F_k telle que $M^{Fk}{}'$=$M^{Fk} \cup$ {A_j}.

Remarque :
Dans le schéma Sh' obtenu, si la dimension D_i ne possède plus de paramètre ($Attribut^{Di}$={}), elle est alors supprimée de l'ensemble des dimensions ; DIM'=DIM-{D_i} et $\forall F' \in FAI'$, $Param'(F')$=$Param(F')$-{D_i}.

EXEMPLE : Nous reprenons l'exemple décrit par le Tableau 25. L'utilisateur transforme le paramètre *region* en mesure. Il réalise une opération de forage vers le bas pour visualiser les départements :

DrillDown(*Sh*, *Geo*, *departement*) = *Sh'*

Puis, il effectue l'opération d'enfoncement suivante :

Push(*Sh'*, *Geo*, *region, Sub*) = *Sh"*

Le tableau suivant illustre le résultat obtenu.

Sub		**Tps**			
		annee	*1998*	*1999*	*2000*
Geo	**departement**	(subventionG, taxe, region)			
	Pyrénées-Atlantique		(140,17, Aq)	(150,19, Aq)	(150,20, Aq)
	Gironde		(140,18, Aq)	(160,21, Aq)	(150,20, Aq)
	PO		(70,8, LR)	(90,11, LR)	(90,11, LR)
	Hérault		(100,15, LR)	(90,14, LR)	(110,19, LR)
	Hte-Garonne		(80,11, MP)	(100,15, MP)	(110,15, MP)
	Tarn		(60,4, MP)	(70,5, MP)	(70,5, MP)
	Hte-Pyrénées		(60,4, MP)	(70,5, MP)	(70,5, MP)
Don.Type = "public"					

Tableau 27 : Résultat de l'enfoncement de l'attribut *region*.

Définition :
Le **retrait** transforme une mesure en paramètre d'une dimension. La syntaxe de l'opérateur est la suivante :

Pull(Sh, F_k, A_j, D_i) = Sh'

Entrées
Sh=(Nom^S, FAI, DIM, $Param$) est un schéma multidimensionnel,

$F_k \in FAI$ est un fait du schéma multidimensionnel,

$A_j \in M^{F1}$ est une mesure du fait courant F_1,

$D_i \in DIM$ est une dimension du schéma multidimensionnel.

Sortie
Sh'=(Nom^S, FAI', DIM', $Param$) est le schéma multidimensionnel dans lequel la dimension D_i' issue de D_i est définie par $D_i'=(Nom^{Di}, Attribut^{Di} \cup \{A_j\}, \{H' \mid \exists H \in H^D, H'=H \cup <A_j>\})$ et $M^{Fk\prime}=M^{Fk}-\{A_j\}$.

EXEMPLE : Nous reprenons l'exemple du schéma multidimensionnel décrit par le Tableau 25. L'utilisateur souhaite transformer la mesure *taxe* en paramètre de la dimension *Geo*. Il réalise donc l'opération d'enfoncement suivante :

Pull(*Sh*, *Sub*, *taxe*, *Geo*) = *Sh'*

Le tableau suivant illustre le résultat obtenu.

Sub			**Tps**			
			annee	*1998*	*1999*	*2000*
Geo	**region**	**taxe**	(subventionG)			
	Aq	*35*		(280)	(null)	(null)
		40		(null)	(310)	(300)
	LR	*23*		(170)	(null)	(null)
		25		(null)	(180)	(null)
		30		(null)	(null)	(200)
	MP	*19*		(200)	(null)	(null)
		25		(null)	(240)	(250)
Don.Type = "public"						

Tableau 28 : Résultat du retrait de la mesure *taxe* dans la dimension *region*.

Remarque :
Ces deux opérations peuvent se généraliser.

- L'opération d'enfoncement peut être appliquée à l'ensemble des paramètres d'une dimension (il s'agit de la **factualisation** de la dimension) ; $Fold(Sh, D_i, F_k) \Leftrightarrow \forall A_j \in Attribut^{Di}, Push(Sh, D_i, A_j, F_k)$.
- L'opération de retrait peut donner lieu à la création d'une nouvelle dimension (il s'agit de la **paramétrisation** d'une mesure) ; $UnFold(Sh, F_k, D_i) \Leftrightarrow Pull(Sh, F_k, A_j, D_i)$ dans le cas particulier où $D_i \notin DIM$.

4.4.5 Opérations de division

L'opération de division totale permet de scinder un schéma multidimensionnel en sous-schémas multidimensionnels comportant un seul fait. Autrement dit, cette opération permet d'obtenir les schémas en étoile contenus dans la constellation de faits du schéma multidimensionnel.

L'intérêt de cette opération est qu'elle permet de fragmenter un schéma multidimensionnel en un ensemble de schémas facilement analysables. Dans le contexte de l'aide à la décision, les utilisateurs peuvent ainsi travailler sur des schémas complexes ou plus simples en fonction de la complexité des données multidimensionnelles contenues dans le magasin.

<u>*Définition*</u> :
La **division totale** scinde un schéma multidimensionnel en plusieurs sous-schémas comportant un seul fait. La syntaxe de l'opérateur est la suivante :

$$\text{TSplit}(Sh) = \{Sh_1, Sh_2, \ldots, Sh_p\}$$

<u>Entrée</u>
$Sh=(Nom^S, FAI, DIM, Param)$ est un schéma multidimensionnel.

<u>Sorties</u>
$\forall i \in [1..p]$, $Sh_i=(Nom^{S'}, FAI', DIM', Param')$ où $Nom^{S'}=Nom^S \cup Nom^{Fi}$, $FAI'=\{F_i\}$, $DIM'=Param(F_i)$ et $Param'(F_i)=Param(F_i)$.

En outre, nous définissons l'opération de division habituelle dans les approches OLAP, qui consiste à fractionner un schéma en plusieurs schémas pour les différentes valeurs d'un paramètre donné.

<u>*Définition*</u> :
La **division** scinde un schéma multidimensionnel en plusieurs sous-schémas obtenus suivant les positions d'un paramètre A_j. La syntaxe de l'opérateur est la suivante :

$$\text{Split}(Sh, D_i, A_j) = \{Sh_1, Sh_2, \ldots, Sh_s\}$$

<u>Entrée</u>
$Sh=(Nom^S, FAI, DIM, Param)$ est un schéma multidimensionnel tel que $FAI=<F>$,

$D_i \in DIM$ est une dimension du schéma multidimensionnel,

$A_j \in Attribut^{Di}$ est un attribut de la dimension D_i.

<u>Sorties</u>
$\forall pos \in Dom(A_j)$, $\forall k \in [1..s]$, les données du schéma Sh_k sont le résultat d'une sélection $\text{Select}(Sh, D_i.A_j=pos)$.

On considère $s=|Dom(A_j)|$ le nombre de positions du paramètre A_j.

4.5 Synthèse

Nous proposons de représenter les données d'un schéma multidimensionnel caractérisant un magasin sous la forme d'un tableau car cette représentation est simple d'utilisation pour les décideurs. En outre, il est très facile de visualiser un tableau de manière graphique (barres, courbes…).

Dans cette section, nous avons défini une algèbre facilitant la manipulation des données multidimensionnelles en proposant différents opérateurs :

- les principaux opérateurs issues de l'algèbre relationnelle sont étendus aux structures multidimensionnelles,
- des opérateurs transforment la granularité des données,
- des opérateurs transforment la structure multidimensionnelle.

L'intérêt de notre algèbre est qu'elle regroupe l'ensemble des opérations proposées dans les différentes approches actuelles (comme le montre le Tableau 6).

	[Li, Wang 1996]	**[Cabbibo, Torlone 1997]**	**[Gyssen, Lakshmanan 1997]]**	**[Agrawal, et al 1997]]**	**[Marcel 1998]]**	**[Lehner, et al 1998] [Lehner 1998]**	**[Pedersen, Jensen 1999]**	**Notre algèbre**
Renommage	x	x	x		x		x	x
Ajout de mesures		x			x			x
Suppression					x			x
Sélection		x	x	x	x	x	x	x
Jointure	x	x	x	x			x	x
Union, Différence, Intersection	x	x	x	x			x	x
Agrégation	x	x	x	x		x	x	x
Roll-up	x	x	x	x	x	x	x	x
Drill-down						x	x	x
Cube			x		x			x
Switch			x					x
Push				x	x			x
Pull				x	x			x
Fold		x	x					x
Unfold		x	x					x
DRotate						x		x
FRotate								x
HRotate								x
Nest						x		x
TSplit								x
Split			x			x		x

Tableau 29 : Comparaison de notre algèbre avec les principaux travaux sur la modélisation multidimensionnelle.

Notre algèbre généralise ces différentes approches car elle s'applique à des schémas en constellation de faits (les autres propositions manipulent des schémas en étoile comportant un fait unique).

Par ailleur, nous avons introduit plusieurs opérateurs nouveaux (*TSplit*, *FRotate*, *HRotate*), inhérents aux structures en constellation de faits qui intègrent des dimensions à hiérarchies multiples.

5 CONCLUSION

Dans ce chapitre, nous avons proposé une solution pour supporter les processus d'analyses. Notre proposition relative à l'approche multidimensionnelle consiste à décrire les magasins de données (dédiés aux analyses OLAP) au travers d'un modèle basé sur les concepts de fait, de dimension et de schéma multidimensionnel (constellation de faits). La contribution de notre modélisation est la généralisation des différents modèles actuellement proposés ; notre modèle repose sur une constellation de faits et de dimensions partagées (ou non partagées) qui supportent des hiérarchies multiples.

Dans un deuxième temps, nous avons défini une démarche de construction des magasins multidimensionnels à partir d'un entrepôt de données conforme au modèle présenté au chapitre II (entrepôt de données historisées et archivées). Cette démarche se décompose en quatre étapes : déterminer le fait, déterminer les dimensions, définir les granularités et définir les hiérarchies de paramètres des dimensions. Il s'agit d'un apport important de notre recherche puisque peu de travaux de recherche proposent actuellement une solution pour transformer les données sous une forme multidimensionnelle.

Enfin, nous avons fourni une algèbre intégrant toutes les opérations multidimensionnelles actuellement proposées. Cette algèbre regroupe des opérateurs permettant :

- d'adapter les opérateurs issues de l'algèbre relationnelle aux structures multidimensionnelles,
- de transformer la structure multidimensionnelle,
- de modifier la granularité des données.

La contribution de notre travail est de proposer une algèbre qui généralise les principaux travaux de recherche des dernières années : notre algèbre supporte l'ensemble des opérations multidimensionnelles habituellement proposées et s'applique sur un schéma multidimensionnel en constellation de faits (il s'agit d'une généralisation du schéma en étoile).

CHAPITRE VI :
REALISATION D'UN OUTIL GRAPHIQUE D'AIDE A LA CONCEPTION D'ENTREPOTS

1 INTRODUCTION A L'ELABORATION GRAPHIQUE D'ENTREPOTS DE DONNEES COMPLEXES ET EVOLUTIVES

1.1 Objectif

Afin de valider le modèle conceptuel dédié aux entrepôts (décrit au chapitre II), nous avons réalisé un prototype d'aide à la conception d'entrepôts de données, intitulé GEDOOH (**G**énérateur d'**E**ntrepôts de **D**onnées **O**rientés **O**bjet et **H**istorisés). Ce chapitre présente GEDOOH, c'est à dire notre démarche et le langage graphique associé au prototype.

Nous proposons un outil reposant sur des représentations graphiques de la source globale et de l'entrepôt. L'utilisation de graphes permet :

- une vision plus explicite de la réalité modélisée que celle offerte par les langages textuels, tabulaires ou iconiques [Canillac 1991] [Le Parc 1997] ;
- un mode d'expression des opérations d'extraction incrémental puisqu'une opération est définie pas en pas en progressant dans le graphe ;
- une représentation de la portée des opérations appliquées sur les graphes.

L'outil GEDOOH est destiné à un ensemble d'administrateurs experts chargés de concevoir et de modéliser un entrepôt de données complexes et évolutives. En effet, l'élaboration d'un entrepôt de données introduit un tel degré de complexité et de compétence (connaissance des informations source disponibles, besoins des décideurs,…) qu'un unique administrateur ne peut maîtriser toute l'information disponible dans l'entreprise ainsi que l'ensemble des besoins du système décisionnel.

1.2 Existant

Les outils issus de l'industrie se concentrent essentiellement sur les aspects de l'interrogation. Plusieurs requêteurs (*Business Objects*, *Impromptu*, *Discover2000*) et outils OLAP (*Powerplay*, *Essbase*, *Oracle Express Analyzer*, *Mercury*) sont proposés. Les requêteurs facilitent la construction de requêtes SQL (jointures masquées, alias de noms d'attributs,…). Les outils OLAP se concentrent sur l'analyse multidimensionnelle en proposant des objets graphiques (tableaux croisés,…). Ces outils utilisent les données de l'entrepôt ou des magasins, mais ne permettent pas leur construction.

Peu d'outils sont disponibles pour assister l'administrateur dans la construction des entrepôts et des magasins de données (*Data Mart Builder*, *Oracle Express Administrator*, *SAS/Warehouse Administrator*, *Warehouse Manager*). Contrairement à GEDOOH, la source et l'entrepôt ne sont pas représentés sous une forme graphique. Ainsi, avec ces outils actuels, le schéma de l'entrepôt doit être construit au préalable ; l'administrateur a la charge d'associer chaque attribut cible de ce schéma à un attribut d'une source de données ; cette phase est complexe, fastidieuse et demande une connaissance importante des structures sources.

De nombreux travaux de recherche ont été consacrés à la définition de langages visuels dans le cadre des bases de données relationnelles (QBE), objets (OOQBE, OHQL, VOHQL) et sémantiques (CANDID, QBD*), mais aucun pour les entrepôts de données complexes et temporelles. La définition d'un langage visuel pour l'élaboration d'entrepôts est d'autant plus justifiée que cette tâche est complexe. Par conséquence, en continuité des travaux réalisés au sein de notre équipe SIG dans le domaine des langages visuels pour la manipulation des bases de données relationnelles (HQL, HQL+, CHQL) et orientées objet (OHQL, VOHQL), nous étudions la définition d'un langage graphique dédié à l'élaboration des entrepôts.

1.3 Proposition

L'outil GEDOOH assiste l'administrateur pour l'élaboration d'entrepôts de données. Le rôle de cet outil est de supporter une construction incrémentale des entrepôts à partir d'une source globale et de produire automatiquement ces entrepôts. L'objectif de cette implantation est de valider les propositions présentées dans ce mémoire de thèse ; plus précisément, GEDOOH traite des propositions décrites au chapitre II (modélisation des entrepôts) et au chapitre III (mécanismes d'extraction).

GEDOOH repose sur des représentations graphiques de la source globale et de l'entrepôt afin d'améliorer la compréhension de l'information modélisée dans la source globale pour l'administrateur. Ce dernier élabore l'entrepôt de données au travers de manipulations exprimées directement sur les graphes. La tâche de l'administrateur en est facilitée. En outre, GEDOOH se caractérise par différents critères :

- Flexible ; la flexibilité implique que le langage graphique utilisé soit adapté aux différentes utilisations possibles.
- Incrémental et réutilisable ; l'administrateur élabore l'entrepôt étape par étape, en visionnant les résultats intermédiaires de chaque opération.
- Uniforme ; l'uniformité impose des modes d'interaction identiques pour les différentes fonctionnalités améliorant l'ergonomie de l'outil.

Ce chapitre comporte trois sections.

- La section 2 présente les caractéristiques générales de GEDOOH, c'est à dire son architecture et les différents modules constituant l'outil.
- La section 3 décrit la démarche de conception d'un entrepôt à partir d'une source globale et définit le langage graphique associé à cette démarche.
- La section 4 définit les règles permettant l'implantation automatique d'un entrepôt dans un système de gestion de bases de données relationnelles.

2 DESCRIPTION GENERALE DE GEDOOH

Le prototype GEDOOH se base sur une approche graphique utilisée à la fois pour représenter la source de données et l'entrepôt de données. De plus, la définition de l'entrepôt est réalisée par des manipulations directes des graphes au travers d'un ensemble d'opérations disponibles.

La Figure 37 décrit l'architecture générale de GEDOOH. Elle comprend pour l'essentiel deux composants : une interface graphique et un module de génération automatique des entrepôts.

- L'**interface graphique** affiche une représentation graphique de la source globale et une représentation graphique de l'entrepôt. L'administrateur définit l'entrepôt à partir de la source globale au travers de cette interface.
- Le **générateur d'entrepôts** consiste en un module de création automatique de scripts. Les scripts générés correspondent à la création de l'entrepôt dans un SGBD hôte ainsi qu'à l'initialisation (première extraction pour peupler l'entrepôt) et aux rafraîchissements (extractions suivantes pour répercuter les évolutions des données source) de l'entrepôt.

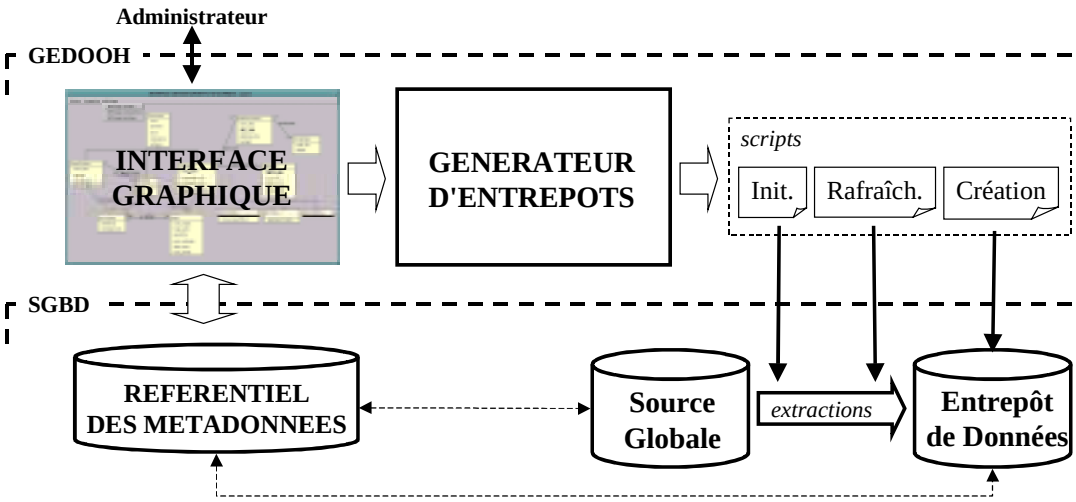


Figure 37 : Architecture du prototype GEDOOH.

Nous détaillons ces deux modules dans les sous-sections suivantes.

2.1 L'interface graphique

Au niveau de l'interface graphique, l'administrateur dispose d'une représentation graphique de la source globale et d'une représentation graphique de l'entrepôt.

2.1.1 La fenêtre de la source globale

La source globale est représentée au travers de son schéma. Il est affiché avec les notations du diagramme des classes d'UML ("*Unified Modeling Language*") [Muller 2000] supportant une sémantique riche et des concepts puissants issus à l'approche objet (tels que l'héritage, l'agrégation…).

La source globale est affichée dans une fenêtre graphique comporte trois zones.

- La *zone 1* est l'intitulé de la fenêtre. Il indique le nom de la source globale qui est affichée.
- La *zone 2* contient les menus permettant à l'administrateur d'agir sur la source globale. Le menu "*Fichiers*" permet d'ouvrir une source globale ou de sauvegarder sa représentation. Le menu "*Affichage*" offre différentes abstractions de représentation (abstraction des compositions, abstraction de l'héritage) [Le Parc 1997]. L'objectif est de faciliter la compréhension du graphe en omettant des détails inutiles ou en affichant le graphe complet. Le menu "*Opérations*" comprend l'ensemble des opérations à effectuer pour

définir une fonction de construction (c'est à dire les opérations de sélection, de projection de masquage, d'augmentation…).

- La *zone 3* affiche le graphe correspondant au schéma de la source globale (diagramme des classes en UML).

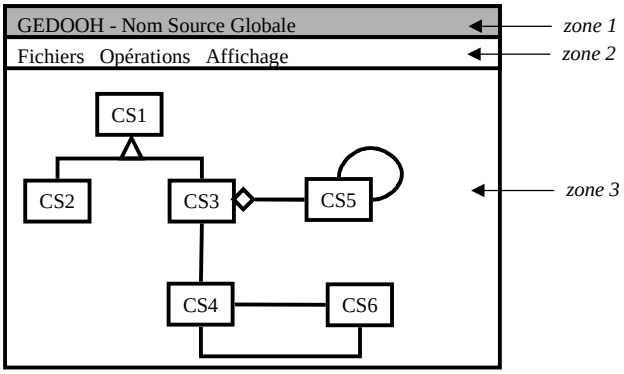


Figure 38 : Fenêtre graphique de la source globale dans GEDOOH.

Le graphe G^{SG} de la source est constitué d'un ensemble de nœuds et de liens. Les nœuds correspondent aux classes d'objets et les liens illustrent les relations entre les classes d'objets (association, composition, héritage) ; seule l'information relative aux classes et à leurs liens est représentée (cette information est suffisante pour permettre à l'administrateur d'appréhender la réalité modélisée).

Le Tableau 30 décrit les notations graphiques (issues des notations du diagramme des classes en UML) adoptées.

Concepts	Notations graphiques
Classes	Notation réduite : **NomClasse** Notation complète : **NomClasse** / NomAttr1:type ... / NomOp1(types):type ...
Associations	**Classe1** — Card2 NomRôle2 / NomRôle1 Card1 — **Classe2**
Compositions	**Classe Composée** ◇ Card2 NomRôle2 / NomRôle1 Card1 **Classe Composante**
Héritages	**Super Classe** ◁ **Sous Classe**

Tableau 30 : Notations graphiques standards pour l'entrepôt et la source globale.

2.1.2 La fenêtre de l'entrepôt

L'entrepôt de données est représenté dans une fenêtre graphique. Les notations utilisées reprennent celles du diagramme des classes UML. Nous étendons les notations pour représenter les concepts spécifiques à notre modélisation tels que les environnements, les filtres des classes…

La fenêtre de l'entrepôt comporte trois zones.

- La *zone 1* est l'intitulé de la fenêtre. Il indique le nom de l'entrepôt qui est affiché.
- La *zone 2* contient les menus permettant à l'administrateur d'agir sur l'entrepôt. Les menus "*Fichiers*" et "*Affichage*" reprennent les fonctionnalités des menus de la fenêtre correspondant à la source globale. Le menu "*Opérations*" comprend l'ensemble des opérations à effectuer pour définir un entrepôt de données, c'est à dire l'élaboration d'une fonction d'extraction, la définition d'un environnement, la spécification de filtres temporels et d'archives,…
- La *zone 3* affiche le graphe correspondant à l'entrepôt (diagramme des classes en UML étendu).

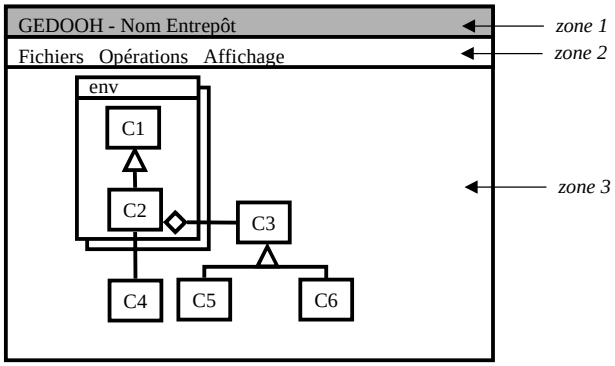


Figure 39 : Fenêtre graphique de l'entrepôt dans GEDOOH.

Le graphe G^{ED} de l'entrepôt est constitué d'un ensemble de nœuds et de liens. Les liens illustrent les relations entre les classes et les nœuds correspondent, soit aux classes, soit aux environnements. Ces derniers ne sont jamais reliés par un lien. Les notations du graphe G^{ED} étendent celles de G^{SG} par des notations dédiées aux environnements, aux filtres des classes et à la taxinomie des attributs, des relations et des opérations.

- Un environnement est représenté par un double rectangle contenant le nom de l'environnement et englobant les classes contenues dans l'environnement.
- Une propriété appartenant au filtre temporel d'une classe est estampillée par un double rectangle blanc tandis qu'un attribut appartenant au filtre d'archives est estampillé par un double rectangle noirci.
- Les noms des éléments dérivés sont préfixés par "D_", les attributs calculés sont préfixés par "C_" et les éléments spécifiques sont préfixés par "S_".

Le Tableau 31 décrit les notations graphiques adoptées.

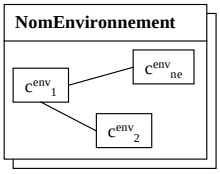
Concepts	Notations graphiques
Environnements	
Filtres temporels	**AttributTemporel**
Filtres d'archives	**AttributArchivé**
Eléments dérivés	**D_nomElement**
Attributs calculés	**C_nomElement**
Eléments spécifiques	**S_nomElement**

Tableau 31 : Notations graphiques spécifiques à l'entrepôt.

2.2 Le générateur automatique

Nous réalisons la génération de l'entrepôt dans un système de gestion de bases de données relationnelles ; le choix d'un SGBD relationnel est motivé par plusieurs raisons.

- La première raison est liée au cadre de nos travaux. Les partenaires du projet REANIMATIC, auquel nous participons, ont opté pour l'offre Oracle v8i. Ce choix est motivé par le coût moins important (que d'autres SGBDs), la stabilité du produit et sa disponibilité sur le marché (par rapport aux SGBDOO peu développés).
- Le second point qui justifie le choix d'un SGBD relationnel est lié aux grandes capacités de stockage ainsi qu'aux performances lors de la manipulation des données. En effet, les systèmes de gestion de bases de données relationnelles offrent d'excellentes performances en terme de rapidité d'accès, de volume de stockage et de stabilité des données.

Le générateur s'appuie sur la description conceptuelle de l'entrepôt ; à partir de la définition orientée objet (UML étendu) de l'entrepôt, le générateur automatique produit des scripts de :

- Création de l'entrepôt. Le script de création définit le schéma de l'entrepôt dans le SGBD hôte, en s'appuyant sur un métaschéma décrivant l'entrepôt (environnements, filtres,…).
- Initialisation de l'entrepôt. Le script d'initialisation consiste en un processus réalisant l'extraction des données source et le stockage de l'information collectée dans le schéma de l'entrepôt.
- Rafraîchissement de l'entrepôt. Le script de rafraîchissement permet de répercuter périodiquement les évolutions de la source au niveau de l'entrepôt. Ces répercutions périodiques tiennent compte de la mise à jour des données courantes, mais également de l'historisation et de l'archivage des données. Ainsi, les scripts de rafraîchissement s'appuient sur trois modules permettant :
  - l'historisation des valeurs dans les états passés,
  - l'archivage des valeurs passées devenues inutiles dans les états archivés,
  - la mise à jour des valeurs des états courants.

2.3 Synthèse

Dans cette section nous avons décrit l'architecture du prototype GEDOOH ; cet outil ce propose d'aider les administrateurs dans leur tâche de conception d'entrepôts de données complexes. GEDOOH comprend :

- une interface graphique qui permet de visualiser la source globale et l'entrepôt de données, et qui permet à l'administrateur de concevoir de manière incrémentale un graphe caractérisant l'entrepôt ;
- un générateur automatique de l'entrepôt qui s'appuie sur le graphe de l'entrepôt pour produire des scripts de création, d'initialisation et de rafraîchissement de l'entrepôt dans un SGBD hôte.

3 LE LANGAGE DE DEFINITION DE L'ENTREPOT

Cette section présente les solutions proposées dans GEDOOH pour définir le graphe d'un entrepôt de données à partir de celui d'une source globale. Nous proposons d'appliquer directement des opérations exprimées au travers de l'interface GEDOOH, facilitant ainsi la tâche de l'administrateur.

3.1 Démarche de la conception

La construction du graphe caractérisant un entrepôt est réalisée à l'aide de l'interface graphique. Après avoir choisi le graphe d'une source globale, la démarche proposée suit trois étapes :

1. **extraction** des données source utiles pour les décideurs et **organisation** de ces données dans l'entrepôt au travers des fonctions de construction (définissant les classes de l'entrepôt),
2. **historisation** des données en définissant des environnements ainsi que les filtres associés aux classes,
3. **configuration** des données par l'intermédiaire de règles qui déterminent le comportement de l'entrepôt (période de rafraîchissement, critères d'archivage…).

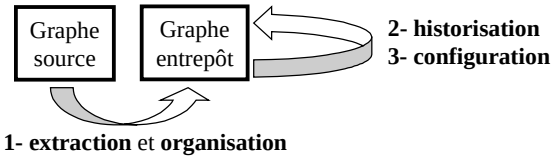


Figure 40 : Démarche générale pour la conception du graphe de l'entrepôt.

3.2 Extraction et organisation

Les classes de l'entrepôt de données (nœuds du graphe) sont élaborées par une fonction de construction. Celle-ci est définie graphiquement au travers d'une succession d'opérations. L'administrateur définit deux types de fonctions :

- des fonctions d'extraction portant sur le graphe source afin de caractériser des données à recopier dans l'entrepôt,
- des fonction de réorganisation portant sur le graphe entrepôt afin de spécialiser la hiérarchie d'héritage de l'entrepôt.

Les fonctions de construction sont exprimées directement sur un graphe (représentant la source de données ou l'entrepôt de données), au travers du langage graphique supporté par GEDOOH. L'administrateur exprime les fonctions de construction au travers d'une requête graphique : il sélectionne des nœuds sur un graphe et déclenche des opérations (à partir des menus "*Opérations*"). Toutes les opérations graphiques sont traduites en opérations algébriques (*cf.* chapitre III) ; ainsi, toute fonction de construction exprimée graphiquement correspond à une suite d'opérations algébriques exécutables.

Le résultat d'une opération est une (ou plusieurs) classe(s) intermédiaire(s) représentée(s) par un ensemble de nœuds et de liens se substituant sur le graphe à ceux sélectionnés. Le résultat est affiché en inverse vidéo pour permettre à l'administrateur de mesurer la portée de son opération, mais également pour lui permettre de poursuivre par une opération suivante. Ainsi, toute fonction exprimée peut servir de base pour exprimer une autre fonction complétée par une opération supplémentaire : une fonction est donc formulée incrémentalement par imbrication d'opérations successives. Ceci permet à l'administrateur d'élaborer des fonctions complexes.

Le système GEDOOH permet de sauvegarder les fonctions (opération "*Sauvegarder*"). La sauvegarde d'une fonction consiste à mémoriser la succession de ses opérations, c'est à dire son processus de construction afin :

- d'éviter de la reformuler si elle est fréquemment utilisée,
- de la compléter ultérieurement si elle est complexe,
- d'utiliser son résultat lors d'opérations ensemblistes (union, intersection, différence).

Lorsque l'administrateur a terminé la définition d'une fonction, il déclenche l'opération "*Finaliser*". Cette opération diffère si la fonction est définie sur le graphe de la source (il s'agit d'une fonction d'extraction) ou sur le graphe de l'entrepôt (il s'agit d'une fonction de réorganisation).

- Sur le graphe de la source, l'opération "*Finaliser*" transforme le graphe de la source dans son état initial (la source est alors de nouveau prête pour accueillir une nouvelle fonction d'extraction). Le graphe de l'entrepôt est complété par le résultat de la fonction d'extraction : une classe entrepôt est créée correspondant à l'extraction des données réalisée par la succession des opérations de la fonction.
- Sur le graphe de l'entrepôt, l'opération "*Finaliser*" transforme le graphe en appliquant la fonction de réorganisation : une super-classe ou sous-classe est générée et les classes initiales sont modifiées conformément à la généralisation ou à la spécialisation.

3.2.1 Elaboration des fonctions d'extraction

3.2.1.1 Projection et masquage

La projection permet de définir les propriétés d'une classe source qui sont dérivées pour générer une classe entrepôt. Inversement, le masquage consiste à définir les propriétés de la classe source qui ne sons pas dérivées ; la classe entrepôt crée est construite à partir des propriétés non spécifiées.

L'expression de la projection (et le masquage) est réalisée au travers d'une fenêtre qui visualise la classe source et qui intègre les différentes cases à cocher.

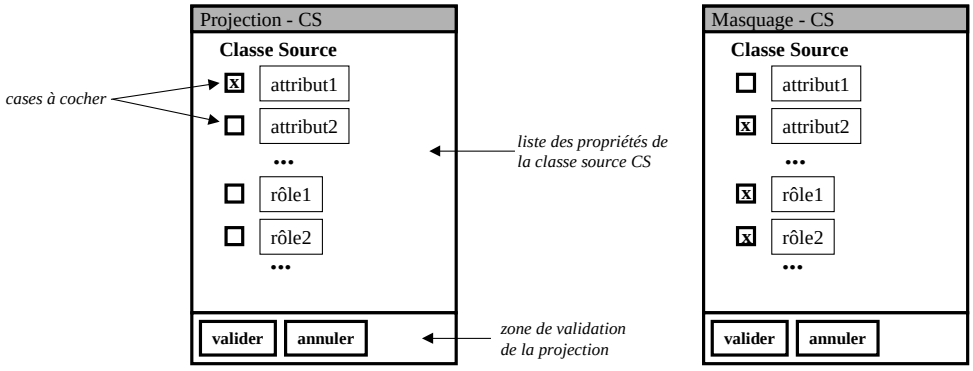


Figure 41 : Expression graphique d'une projection (ou d'un masquage).

3.2.1.2 Sélection

L'opération de sélection permet de restreindre l'ensemble des objets d'une classe source à partir de laquelle est générée une classe entrepôt.

L'expression d'une sélection est réalisée à l'aide de deux fenêtres. Une première fenêtre permet de saisir le prédicat de la sélection. Une seconde fenêtre visualise, au fur et à mesure de sa construction, le prédicat exprimé sous la forme d'un arbre.

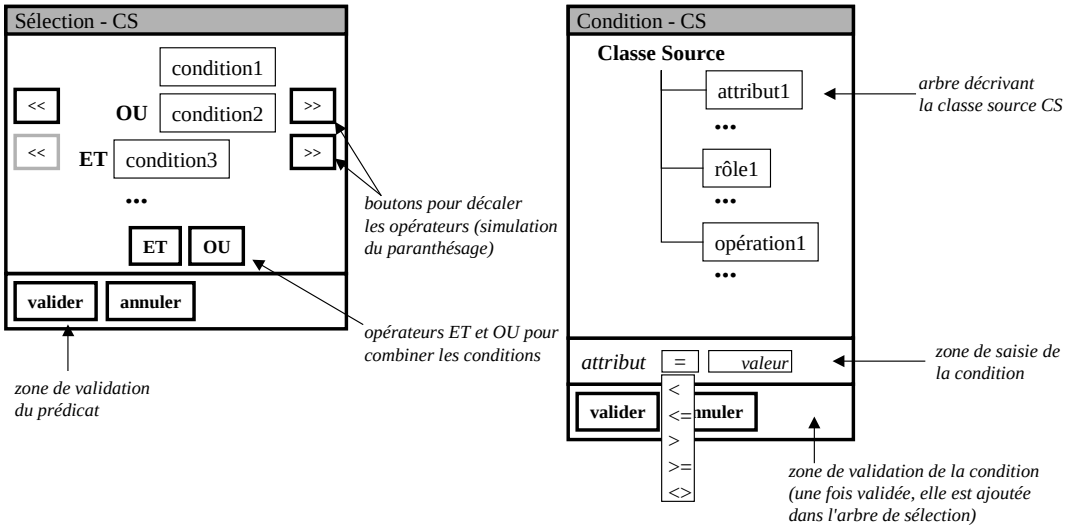


Figure 42 : Expression graphique d'une sélection.

Un prédicat est composé de conditions (prédicats simples) liées par les opérateurs logiques "*et*" et "*ou*". La saisie d'une condition s'effectue en trois étapes :

- premièrement, à partir d'un arbre représentant le type de la classe source (attributs, relations et opérations), l'administrateur choisit l'élément sur lequel porte la condition ;
- deuxièmement, après avoir choisi l'élément impliqué dans la condition, l'administrateur désigne un opérateur parmi une liste proposée par GEDOOH (suivant le type de l'élément) ;
- troisièmement, l'administrateur saisit une valeur de comparaison.

L'administrateur valide la condition. Le prédicat de sélection apparaît dans une seconde fenêtre sous la forme d'un arbre au fur et à mesure de sa construction. L'administrateur choisit un opérateur logique pour saisir chaque condition qui suit la première. Des boutons de décalage permettent de simuler graphiquement l'imbrication des conditions avec parenthésage ; par exemple, dans la figure précédente, on exprime le prédicat *(condition1 OU condition 2) ET condition3…*

Remarque :
Le prédicat de sélection peut porter sur des objets source liés. Pour exprimer un tel prédicat, une jointure implicite doit être utilisée afin d'accéder, à partir d'un objet, aux objets liés. Les jointures implicites sont induites par les liens de composition et d'association du modèle. Dans GEDOOH, elles sont exprimées à partir de l'arbre représentant le type de la classe sur laquelle l'opération de sélection a été déclenchée.

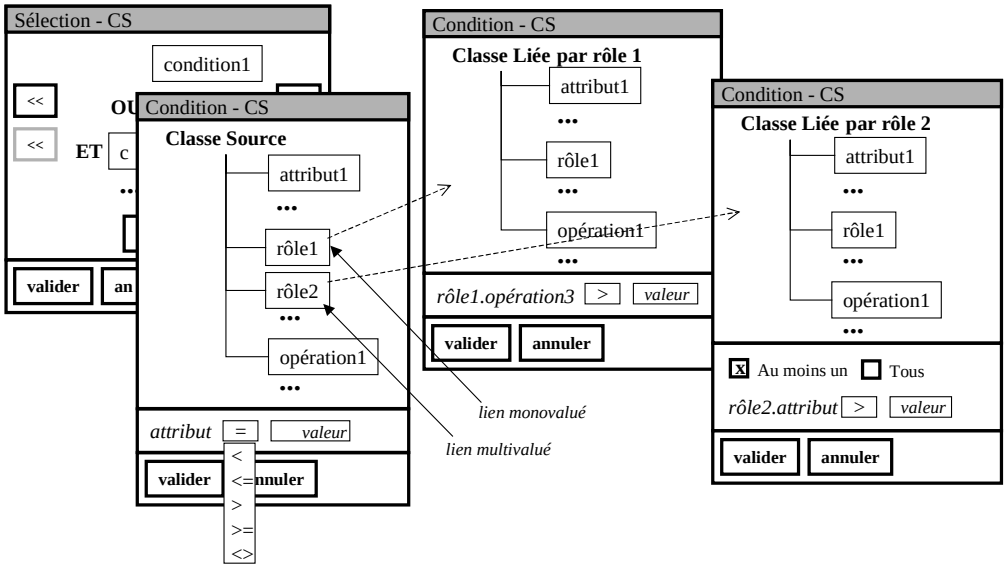


Figure 43 : Expression graphique d'une sélection sur les objets liés (jointure implicite).

Une fenêtre permet d'indiquer les conditions que doivent vérifier les objets liés. Le prédicat sur les objets liés n'a aucune incidence sur la classe à laquelle ces objets appartiennent. Il permet uniquement de restreindre l'ensemble des objets de la classe de départ en se basant sur leurs objets liés.

3.2.1.3 Jointure

L'opération de jointure permet de définir une classe entrepôt à partir de deux classes source entre lesquelles un lien est établi, ce lien n'étant pas modélisé explicitement par une relation d'association ou de composition (cas des jointures implicites).

L'expression de la jointure est réalisée directement sur l'une des deux classes ; l'administrateur désigne ensuite l'autre classe directement sur le graphe. Une fenêtre permettant la saisie du prédicat est ensuite disponible.

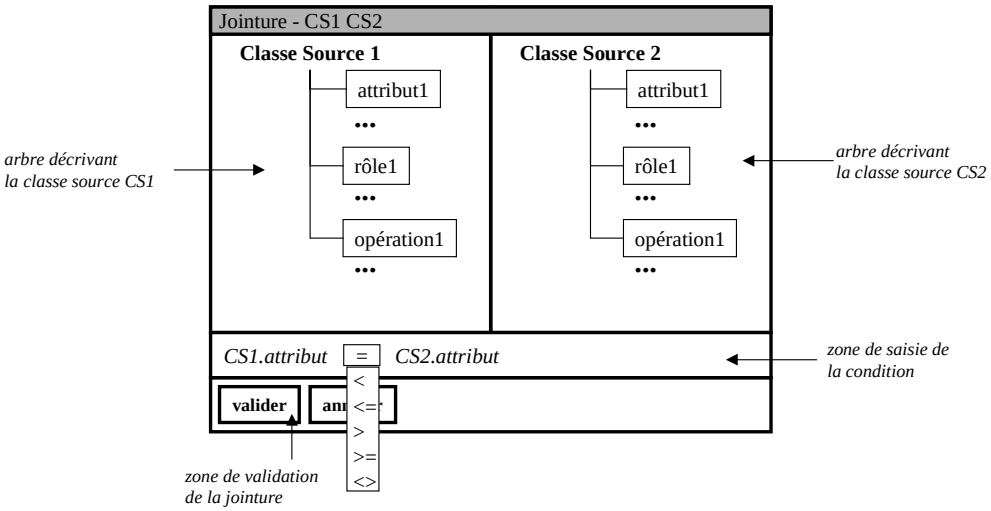


Figure 44 : Expression graphique d'une jointure.

La fenêtre est composée de plusieurs zones :

- une zone décrit le schéma de la première classe source sous forme d'un arbre,
- une zone décrit le schéma de la seconde classe source sous forme d'un arbre,
- une zone correspond à l'expression de la condition de jointure. Lorsqu'un attribut ou une opération a été choisi dans l'un des arbres, tous les attributs et les opérations de l'autre arbre ne pouvant convenir deviennent inaccessibles (garantissant ainsi la correction syntaxique et sémantique des conditions de jointure).

3.2.1.4 Groupement et dégroupement

Le groupement consiste à regrouper les objets source, à partir desquels la classe entrepôt est construite, en fonction d'un ou plusieurs attributs de regroupement. Le dégroupement réalise l'opération contraire en dupliquant les objets source en fonction des valeurs d'un attribut multivalué (il ne s'agit pas de l'opération inverse).

L'expression du groupement (et du dégroupement) est réalisée au travers d'une fenêtre graphique qui visualise la classe source. L'administrateur visualise l'ensemble des propriétés (attributs et relations) de la classe source.

- Pour une opération de groupement, il indique les propriétés de regroupement. Puis l'administrateur saisit le nom de l'attribut de groupement.
- Pour le dégroupement, seules les propriétés multivaluées, dont les valeurs sont susceptibles d'être dégroupées, sont affichées. L'administrateur indique les propriétés de dégroupement.

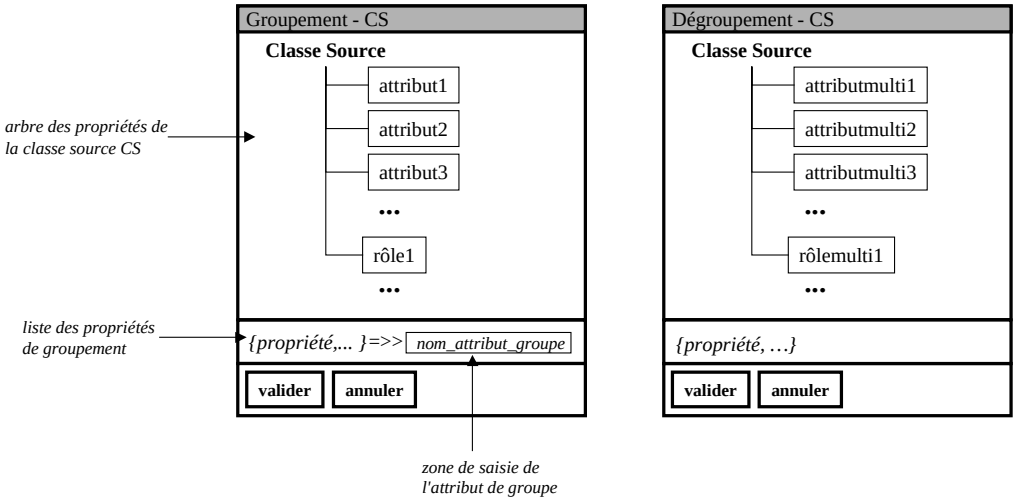


Figure 45 : Expression graphique d'un groupement (ou un dégroupement).

3.2.1.5 Union, intersection et différence

Les opérations ensemblistes d'union, d'intersection et de différence permettent de combiner deux classes source pour générer une classe entrepôt.

L'expression d'une opération ensembliste est réalisée au travers d'une fenêtre en indiquant l'une des deux classes source impliquées. L'administrateur précise l'opération ensembliste désirée. Puis il indique la classe constituant le deuxième opérande ; il s'agit de choisir parmi la liste des classes intermédiaires calculées par une autre opération. Seules sont proposées les classes intermédiaires ayant le même type que la première classe.

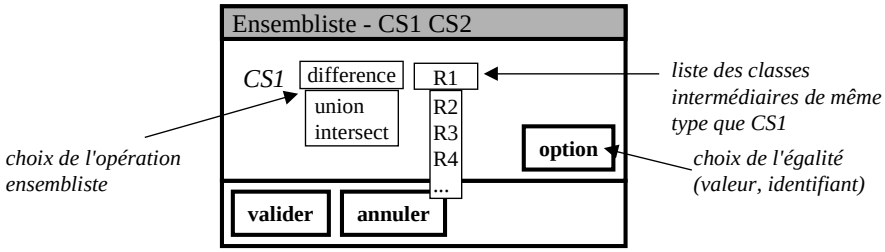


Figure 46 : Expression graphique d'une union (ou d'une intersection ou d'une différence).

3.2.1.6 Augmentation

L'augmentation consiste à créer des propriétés spécifiques ou des attributs calculés sur une classe source à partir de laquelle la classe entrepôt est générée.

L'expression de l'augmentation est réalisée au travers d'une ou plusieurs fenêtres graphiques selon qu'il s'agisse d'ajouter d'un attribut calculé ou une propriété spécifique.

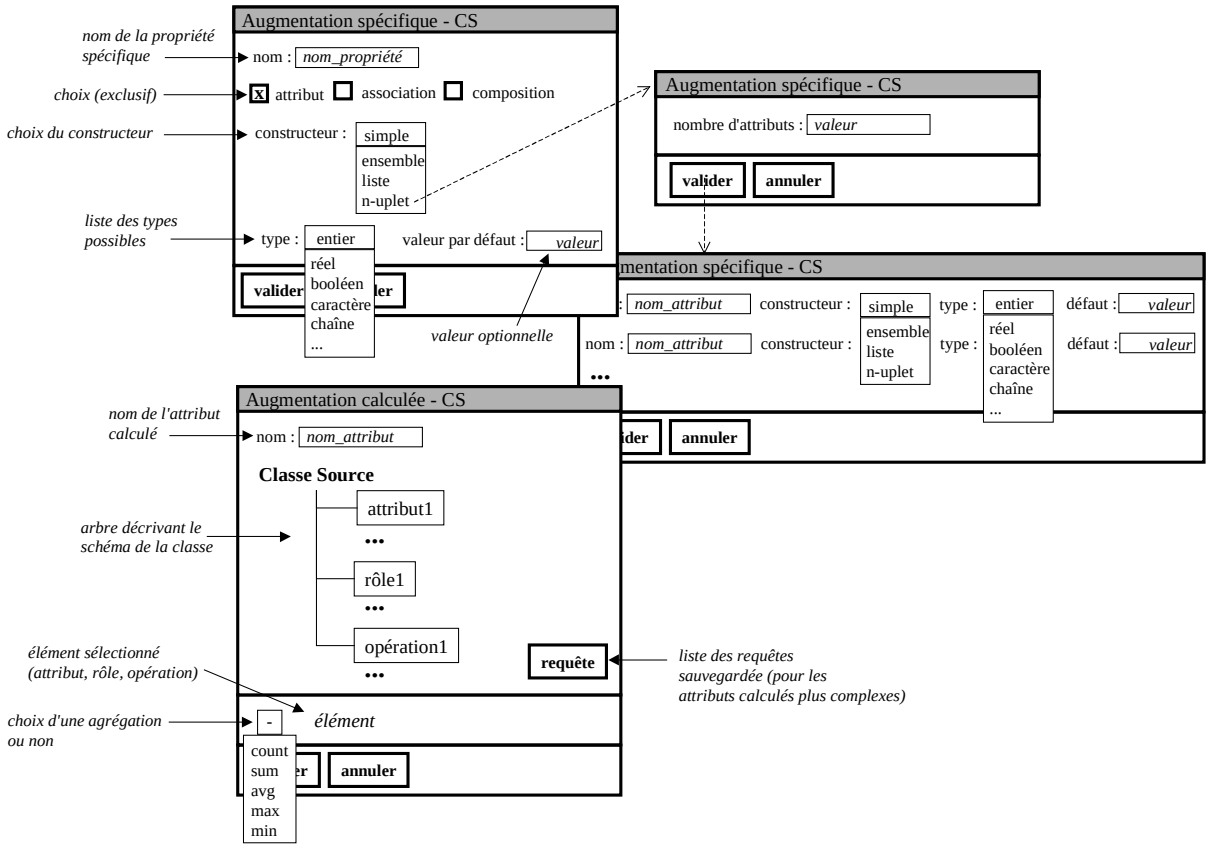


Figure 47 : Expression graphique d'une augmentation spécifique (ou calculée).

Lorsque l'administrateur ajoute une propriété spécifique, il précise le nom de la propriété, puis il indique s'il s'agit d'un attribut, d'une association ou d'une composition. La liste des constructeurs et des types possibles sont déterminées par le système. Si l'administrateur choisit le constructeur n-uplet (pour un attribut), une nouvelle fenêtre permet de spécifier le nombre d'attributs, puis une autre fenêtre autorise la saisie de chaque attribut (contenu dans le n-uplet).

Lorsque l'administrateur ajoute un attribut calculé, il saisit le nom de l'attribut, puis plusieurs possibilités sont offertes.

- Soit l'administrateur sélectionne un élément (attribut, relation ou opération) du type de la classe. Le résultat sera calculé directement suivant la valeur retournée par l'élément ou bien en appliquant une agrégation sur un ensemble de valeurs retournées.
- Soit l'administrateur désire exprimer un calcul complexe au travers d'une autre fonction de construction (requête). Une agrégation peut également être appliquée sur les valeurs retournées par la fonction choisie.

3.2.2 Finalisation des fonctions d'extraction

L'administrateur définit le graphe de l'entrepôt en construisant des fonctions d'extraction. Ces dernières sont formées d'une succession d'opérations appliquées sur le graphe source.

L'opération "*Finaliser*" consiste pour le système GEDOOH à appliquer le processus d'extraction qui permet de définir une classe entrepôt. Ce processus correspond à une

expression algébrique exprimée dans l'algèbre présentée au chapitre III. En outre, lorsqu'une fonction est finalisée, le système traite de automatiquement :

- la redéfinition des liens entre les nœuds du graphe de l'entrepôt,
- la reconstruction des hiérarchies existantes (hiérarchie d'héritage, hiérarchie de composition).

Ces deux étapes sont réalisées de manière semi-automatique par des interactions avec l'administrateur.

3.2.2.1 Redéfinition des liens

Pour redéfinir les liens projetés entre les classes de l'entrepôt, l'administrateur doit spécifier, pour chaque relation projetée, à quelle classe entrepôt correspond les objets liés de la source. Pour chaque rôle projeté, le système propose automatiquement les classes représentant dans l'entrepôt la classe source liée. L'administrateur spécifie, pour chaque rôle projeté, la (ou les) classe(s) entrepôt liée(s).

3.2.2.2 Reconstruction des hiérarchies existantes

Pour la reconstruction des hiérarchies existantes, l'administrateur peut indiquer au système, par le menu "*Opérations*" de la fenêtre du graphe source, s'il souhaite conserver ou non les hiérarchies d'héritage et les hiérarchies de composition des classes source impliquées dans une fonction d'extraction ; l'administrateur dispose d'une option à cocher (dans le menu "*Opérations*") pour spécifier la conservation de ces hiérarchies.

La reconstruction des hiérarchies est réalisée par une interaction au travers d'une fenêtre relative à la hiérarchie d'héritage et d'une fenêtre relative à la hiérarchie de composition.

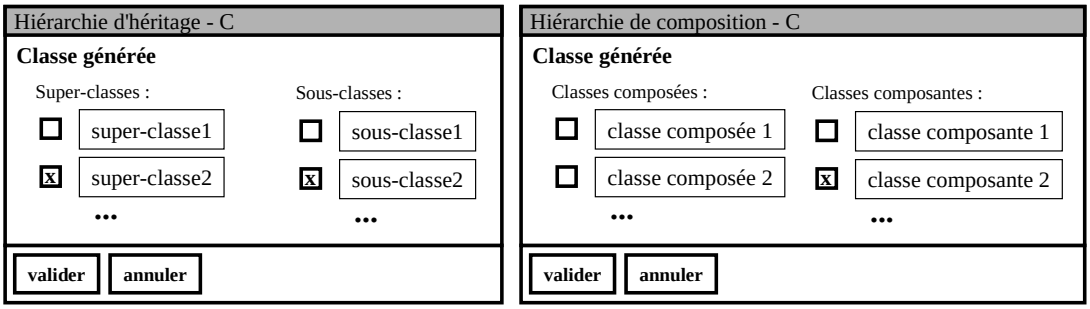


Figure 48 : Principe de l'interaction pour traiter les hiérarchies d'héritage et de composition existantes.

En fonction des options choisies, le système réalise ou non, lors de l'exécution d'une fonction d'extraction, cette interaction supplémentaire avec l'administrateur au cours de laquelle l'administrateur indique les super-classes, les sous-classes, les classes composées et les classes composantes conservées.

Plusieurs problèmes sont résolus par le système de manière automatique.

- Chaque classe source recopiée est limitée aux structures projetées et aux objets collectés. Autrement dit, une classe source automatiquement recopiée dans l'entrepôt (pour reconstruire un lien d'héritage ou de composition) est construite à partir des seules propriétés et opérations intervenant dans la construction de la classe entrepôt générée.

- Lorsqu'une même classe source est recopiée automatiquement plusieurs fois dans l'entrepôt suite à différentes fonctions d'extraction, le système détermine automatiquement les éléments communs entre chaque recopies et construit les classes intermédiaires éventuellement nécessaires.

3.2.3 Elaboration des fonctions de réorganisation

L'administrateur peut adapter le graphe de l'entrepôt en fonction de ses besoins spécifiques. Pour cela, il dispose d'opérations permettant de supprimer des classes, des liens, des attributs , des opérations mais également de créer des super-classes et des sous-classes. Nous ne détaillons que les fonctionnalités complexes de généralisation et de spécialisation.

Cette étape diffère de la phase de reconstruction des hiérarchies existantes présentée précédemment. Il s'agit dans ce cas pour l'administrateur de créer de nouvelles super-classes et sous-classes ; ces classes sont construites à partir d'une fonction de réorganisation exprimée par une succession d'opérations de réorganisations appliquées sur le graphe de l'entrepôt. Ces opérations correspondent aux opérations de réorganisation décrites au chapitre III.

3.2.3.1 Généralisation

La généralisation consiste à ajouter un nœud dans le graphe de l'entrepôt et des liens d'héritage (qui partent du nouveau nœud) en modifiant certains nœuds ; il s'agit de créer une super-classe entrepôt.

L'expression de la généralisation est réalisée au travers d'une fenêtre graphique. L'administrateur déclenche la généralisation à partir d'une classe entrepôt. Il précise ensuite les autres classes entrepôt participant à la généralisation.

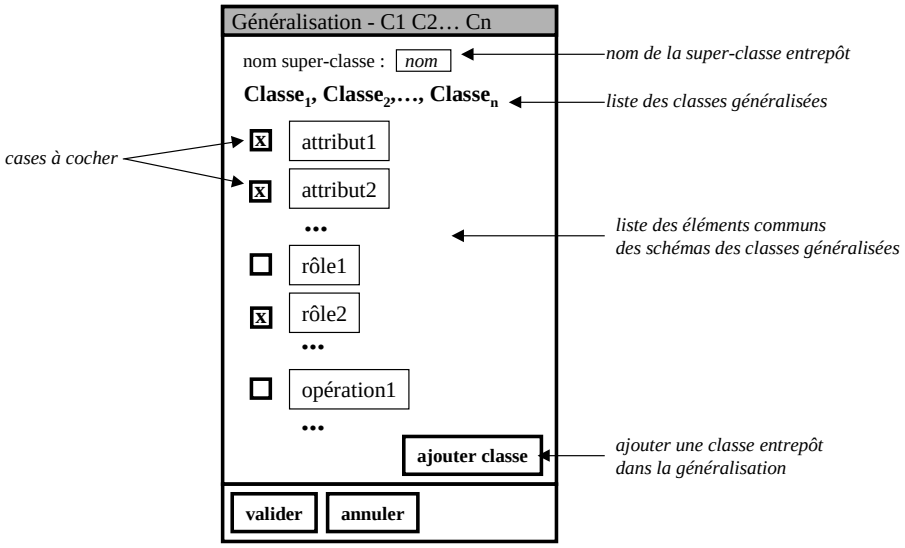


Figure 49 : Expression graphique d'une généralisation.

L'administrateur saisit le nom de la super-classe créée, puis il coche les éléments (attributs, relations, opérations) placés dans la super-classe, parmi les éléments communs aux types des classes à généraliser.

3.2.3.2 Spécialisation

La spécialisation consiste à ajouter un nœud dans le graphe de l'entrepôt et des liens d'héritage (qui atteignent le nouveau nœud) en modifiant certains nœuds ; il s'agit de créer une sous-classe entrepôt.

L'expression de la spécialisation est réalisée au travers d'une fenêtre graphique. L'administrateur déclenche la spécialisation à partir d'une classe entrepôt. Il précise ensuite les autres classes entrepôt participant à la spécialisation.

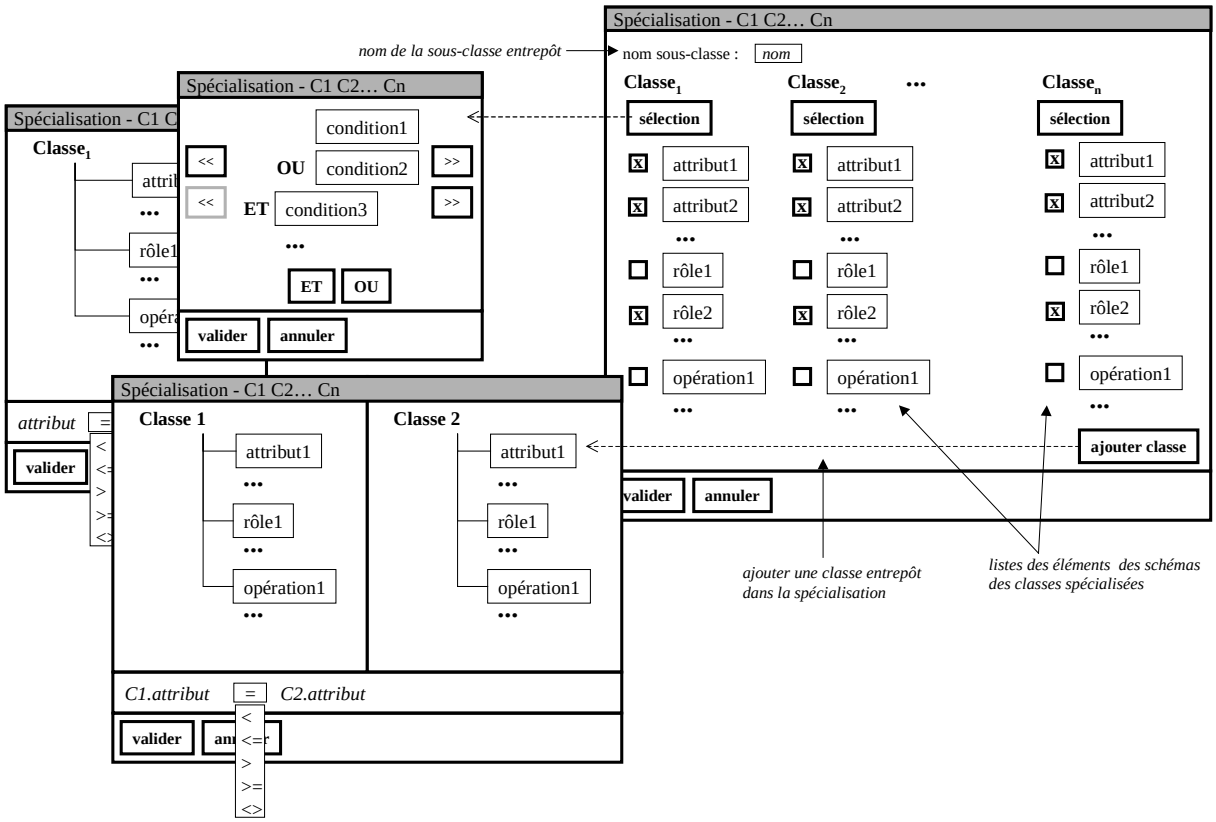


Figure 50 : Expression graphique d'une spécialisation.

L'administrateur saisit le nom de la sous-classe, puis il indique les éléments (attributs, relations, opérations) placés dans la sous-classe. Lorsque plusieurs classes participent à la spécialisation, les classes sont liées par des conditions de jointure lors de leur ajout.

L'administrateur peut spécifier un prédicat de sélection afin de préciser l'ensemble des objets entrepôt qui sont placés dans la sous-classe. Dans ce cas, l'arbre des éléments dans la sélection de la classe *i* est réduit aux éléments en commun.

3.2.4 Finalisation des fonctions de réorganisation

Pour les fonctions de réorganisation, l'opération "*Finaliser*" consiste pour le système GEDOOH à appliquer le processus qui permet de réorganiser la hiérarchie d'héritage en créant une sous-classe ou une super-classe. Ce processus correspond à une expression algébrique exprimée dans l'algèbre présentée au chapitre III.

3.2.5 Exemple complet d'extractions et de réorganisation

Dans cette section, nous décrivons l'exemple complet de construction de l'entrepôt de données (décrit au chapitre II) à partir de la source globale (décrite en annexe A). Il s'agit des fonctions de construction présentées algébriquement au chapitre III.

EXEMPLE : L'administrateur souhaite élaborer cet entrepôt de données à partir de la source globale de données médicales.

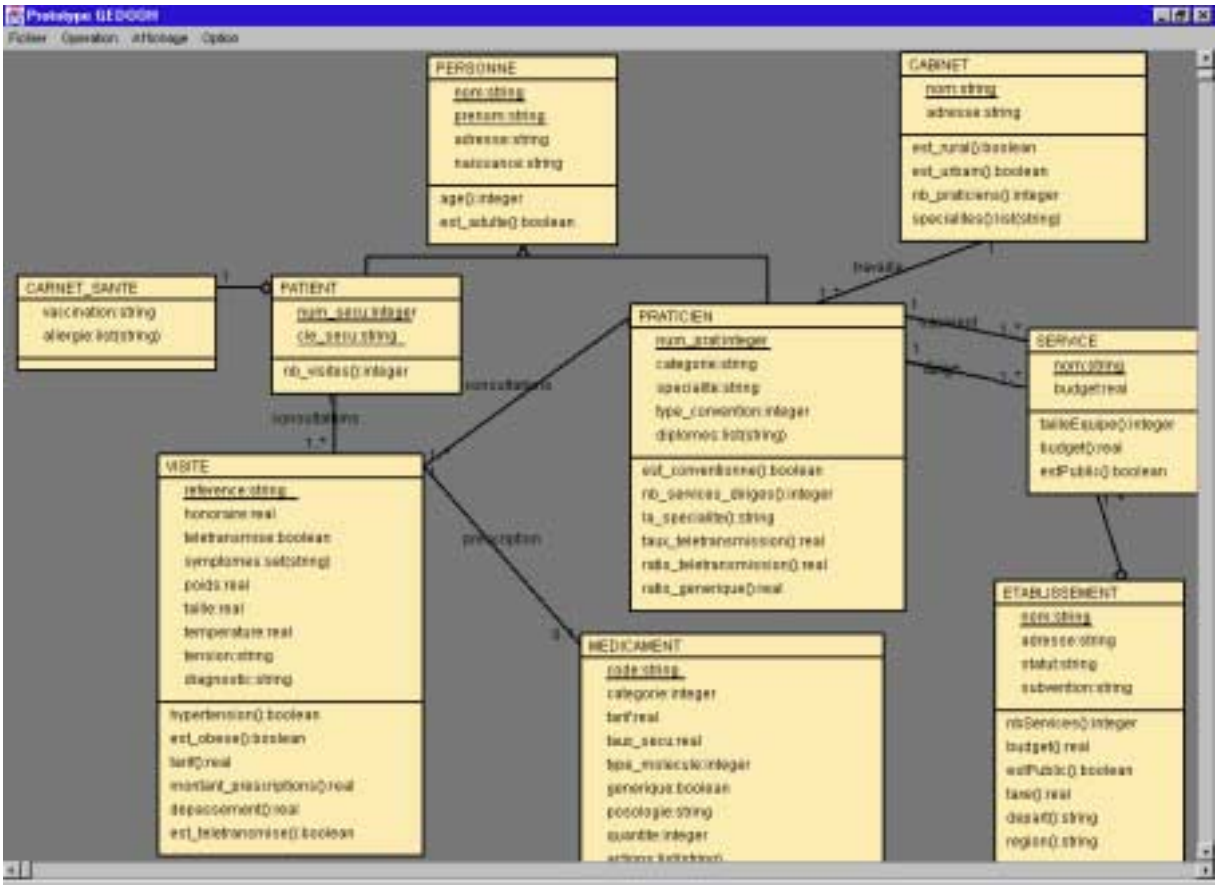


Figure 51 : Représentation de la source globale de données médicales.

L'administrateur élabore l'entrepôt en définissant les fonctions de construction permettant d'extraire :

-les établissements,

-les services qui composent les établissements,

-les spécialistes en conservant la hiérarchie d'héritage source,

-leurs cabinets médicaux des spécialistes,

-les visites effectuées par les patients chez des spécialistes ainsi que des prescriptions ordonnées à cette occasion par les professionnels de santé.

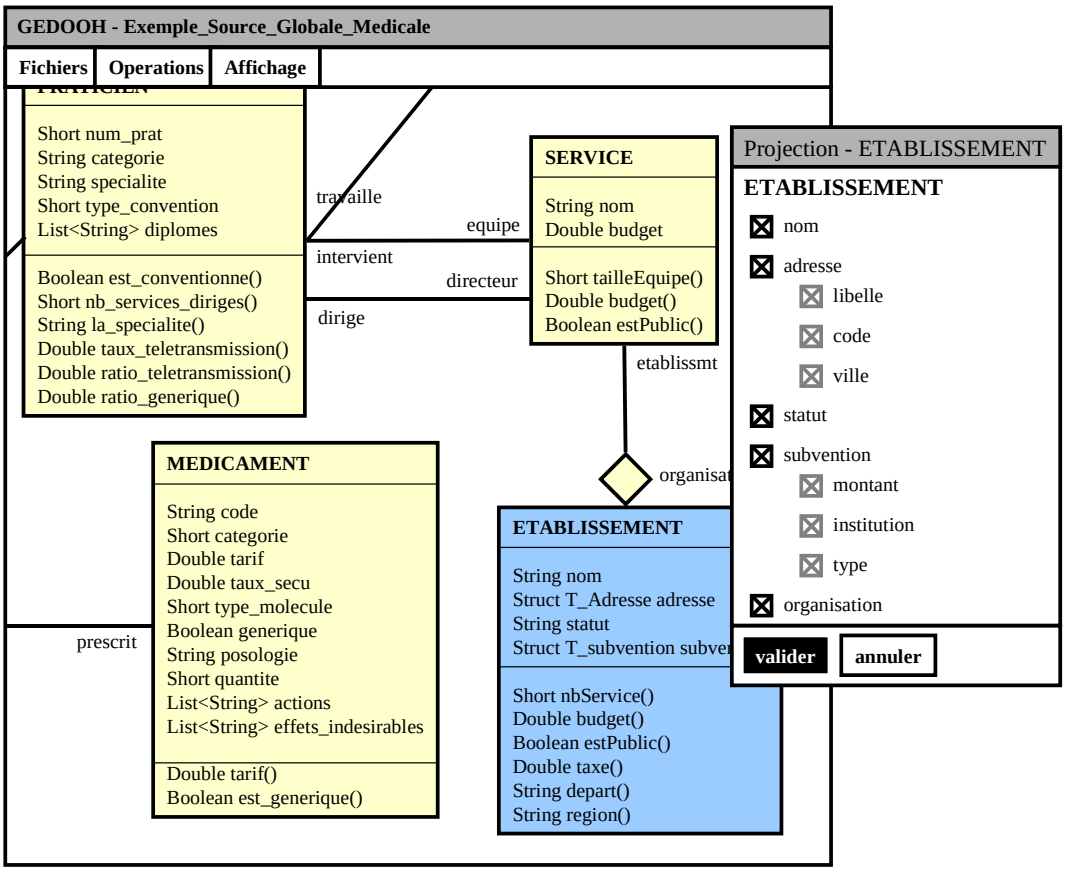


Figure 52 : Elaboration de la fonction d'extraction des établissements.

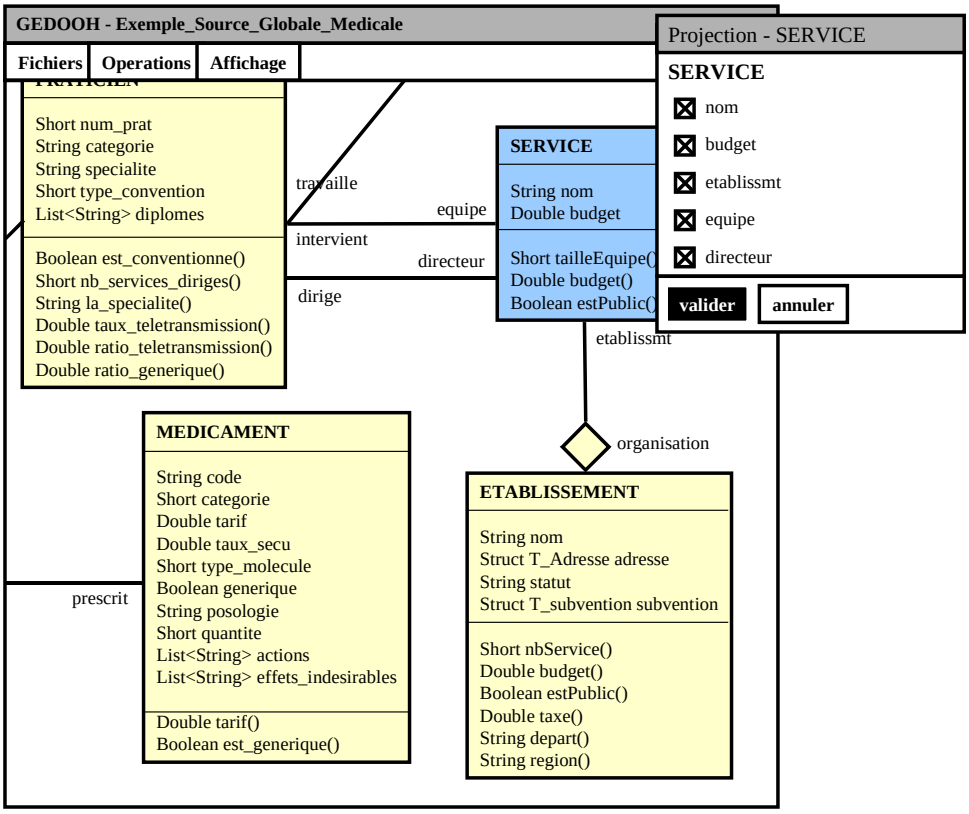


Figure 53 : Elaboration de la fonction d'extraction des services.

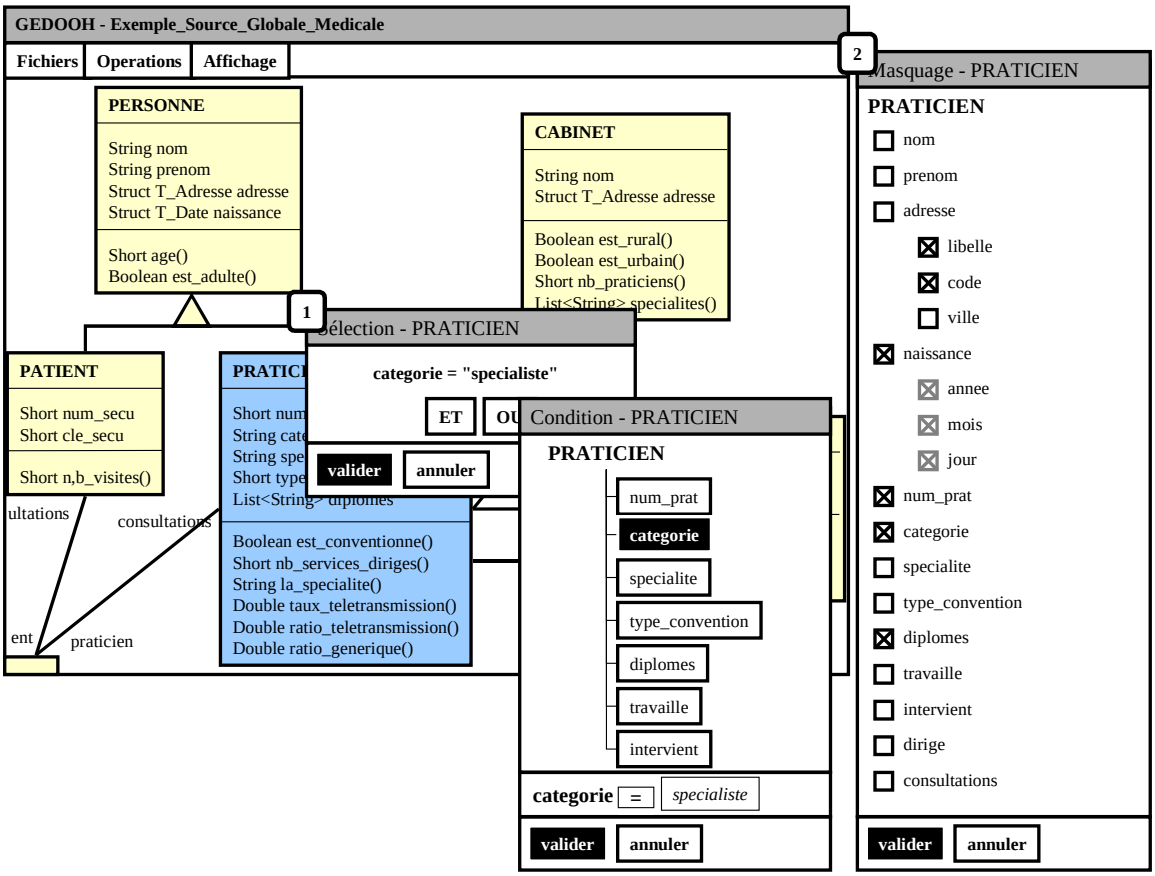


Figure 54 : Elaboration de la fonction d'extraction des praticiens.

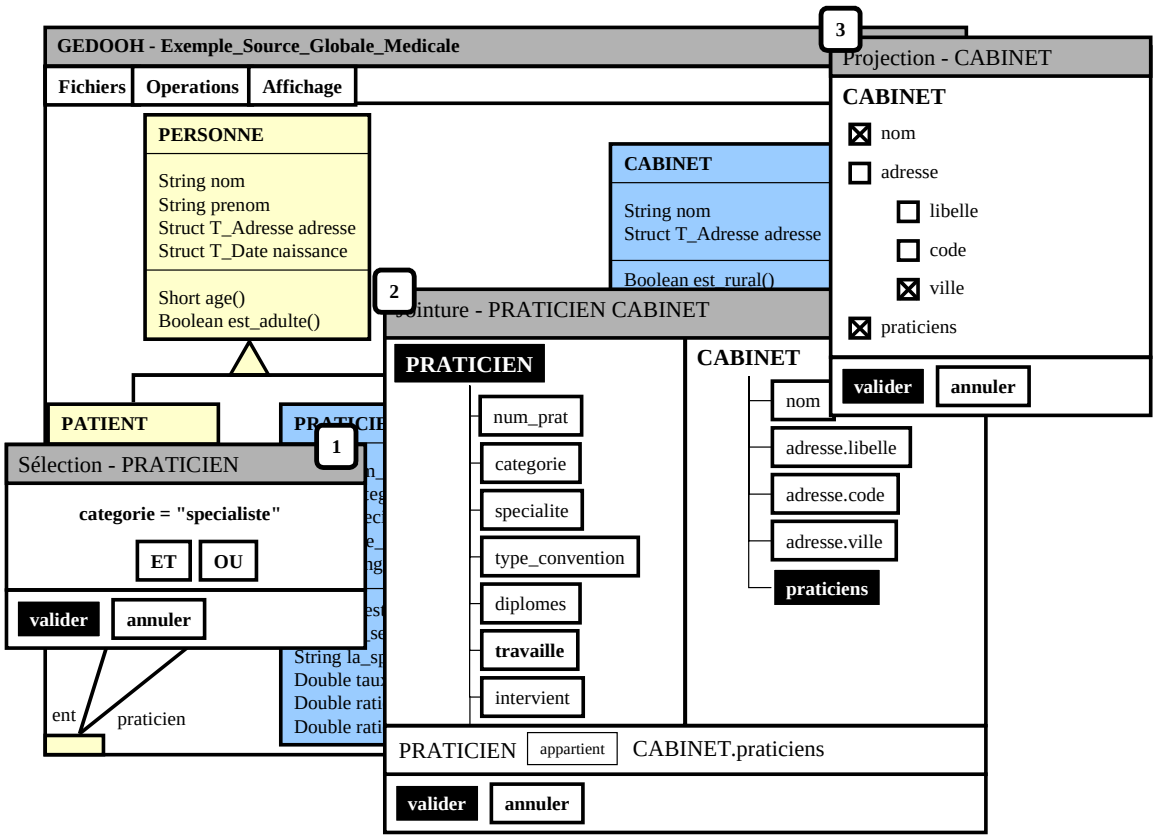


Figure 55 : Elaboration de la fonction d'extraction des cabinets médicaux.

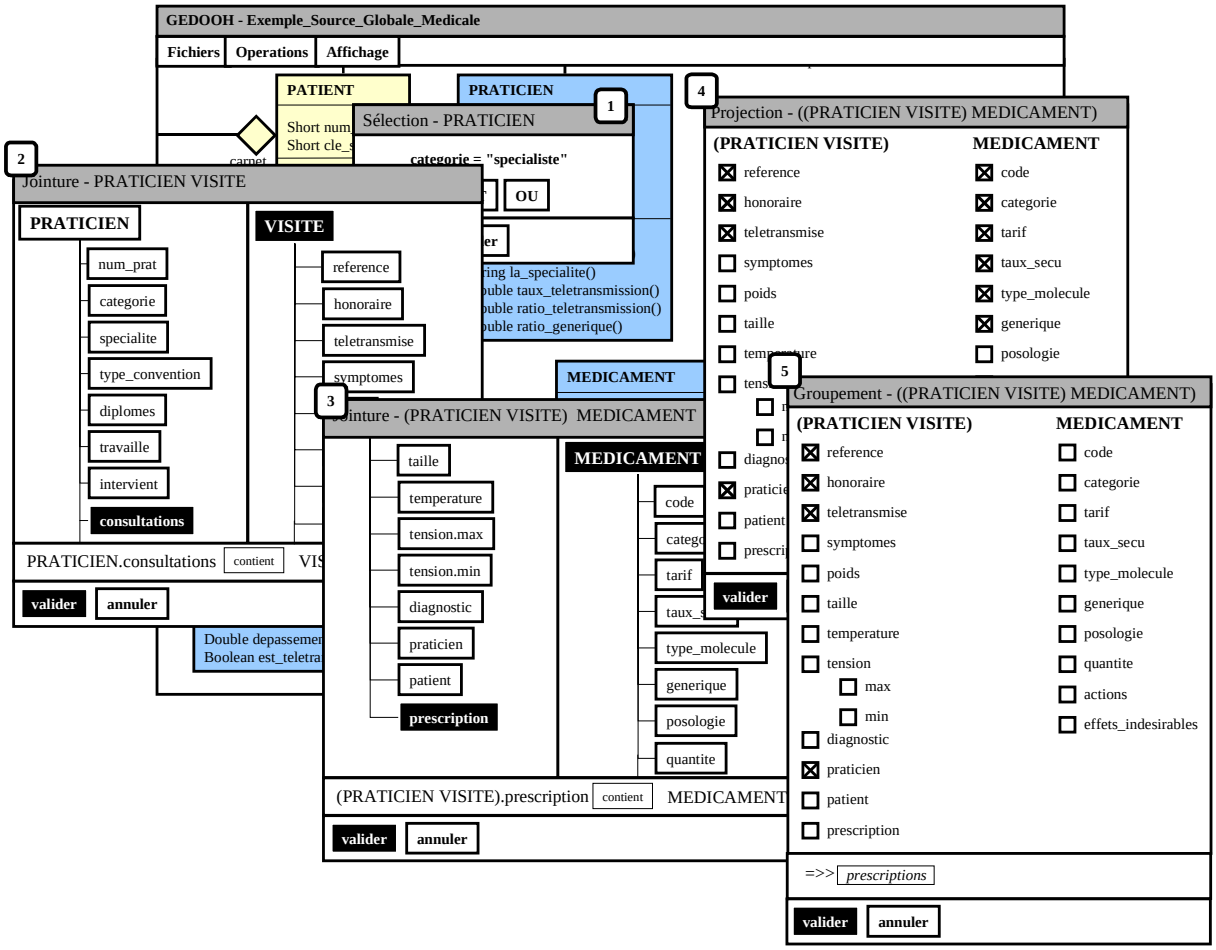


Figure 56 : Elaboration de la fonction d'extraction des visites.

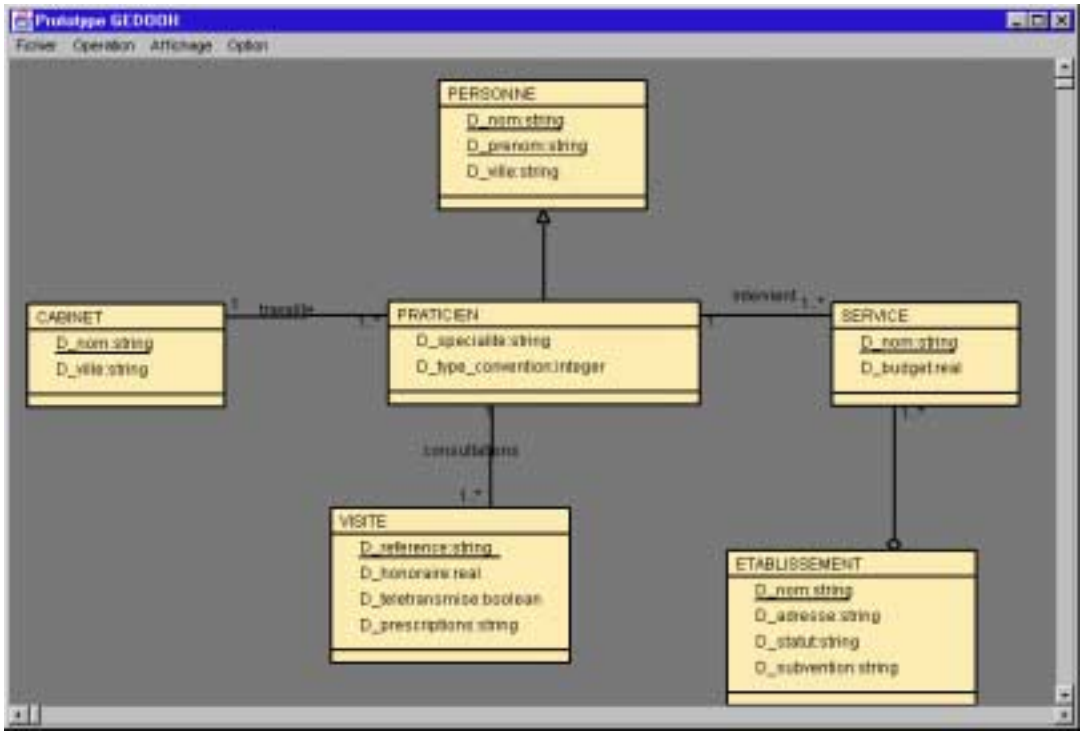


Figure 57 : Représentation de l'entrepôt de données obtenu.

Après élaboration de chacune des fonctions, l'administrateur finalise celles-ci : une classe entrepôt est créée dans la fenêtre de l'entrepôt et le graphe de la source globale est remis dans son état initial pour construire une nouvelle fonction.

La Figure 57 présente le graphe de l'entrepôt finalement obtenu.

3.3 Historisation

La seconde étape de notre démarche consiste pour l'administrateur à se focaliser sur les aspects temporels de l'entrepôt. L'administrateur définit dans cette étape les différents environnements de l'entrepôt, et il spécifie les filtres associés à chaque classe contenue dans un environnement (seules les classes des environnements sont historisées et archivées).

3.3.1 Définition graphique des environnements

Pour définir graphiquement une partie historisée, l'administrateur sélectionne directement sur le graphe de l'entrepôt les classes qu'il souhaite placer dans un environnement. Après avoir déclenché l'opération de création d'un environnement, il spécifie ensuite le nom de celui-ci.

Le système construit alors un graphe résultat reprenant le graphe initial, augmenté par un nœud supplémentaire représentant l'environnement et englobant les classes de ce dernier suivant les notations introduites à la section 2.1.2.

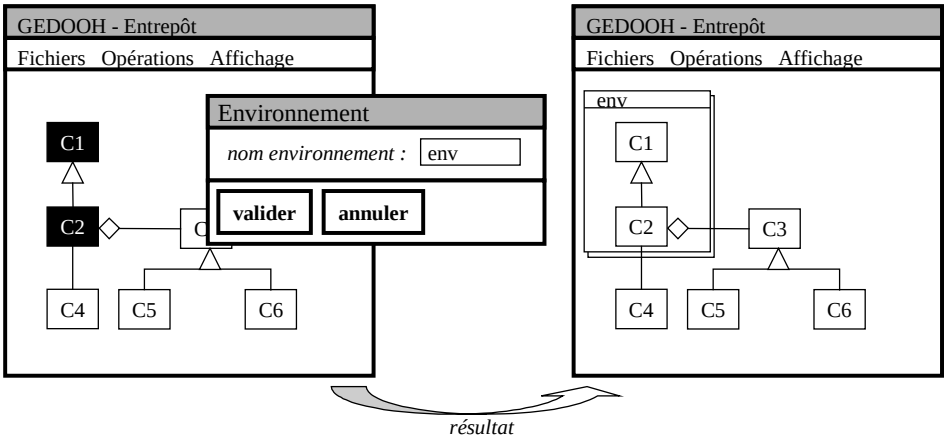


Figure 58 : Création d'un environnement.

Le système vérifie que les environnements soient disjoints (une classe appartient au plus à un environnement).

3.3.2 Définition graphique des filtres temporels

Pour définir un filtre temporel, l'administrateur peut sélectionner une classe du graphe de l'entrepôt contenue dans un environnement. Cette opération consiste à spécifier la structure des états passés qui conservent les évolutions détaillées des objets de la classe.

L'expression d'un filtre temporel est réalisée au travers d'une fenêtre graphique dans laquelle l'administrateur coche les propriétés ou les opérations contenues dans le filtre temporel.

Le système propose l'ensemble des éléments (attributs, relations, opérations) de la classe entrepôt. L'administrateur coche les éléments historisés. Seuls les rôles reliant les classes du même environnement sont proposés car les autres rôles ne sont pas historisables.

Les opérations ne sont pas historisées en tant que telles, mais les opérations retournant un résultat peuvent être historisées. Il s'agit en fait de conserver la valeur du résultat de l'opération sous la forme d'un attribut de l'état passé : pour cela, l'administrateur place l'opération à historiser dans le filtre temporel.

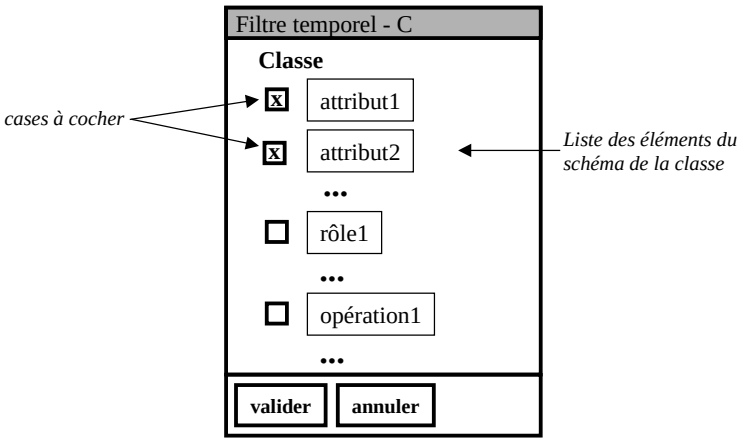


Figure 59 : Définition d'un filtre temporel.

3.3.3 Définition graphique des filtres d'archives

Pour définir un filtre d'archives, l'administrateur peut sélectionner une classe contenue dans un environnement et dont le filtre temporel n'est pas vide. Cette opération consiste à spécifier la structure des états archivés qui conservent un résumé des évolutions des objets de la classe.

L'expression d'un filtre d'archives est défini au travers d'une fenêtre graphique dans laquelle l'administrateur coche les attributs contenus dans le filtre d'archives.

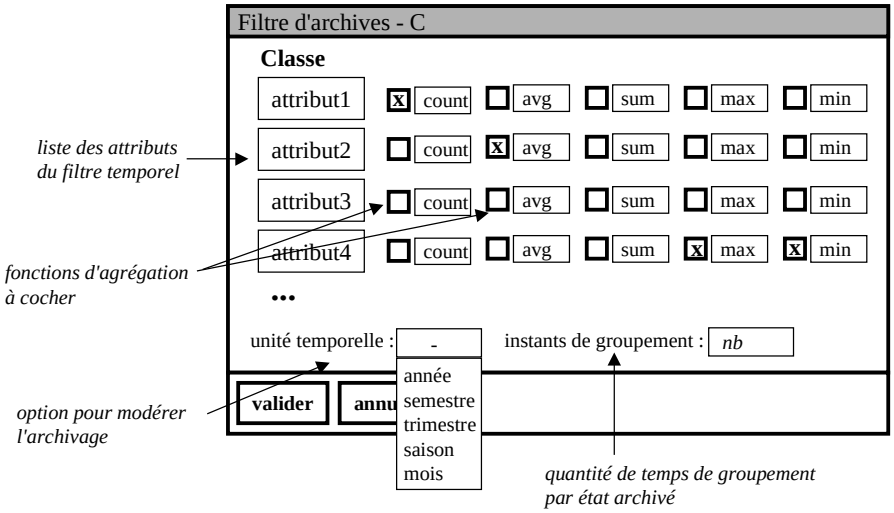


Figure 60 : Définition d'un filtre d'archives.

Pour chaque attribut proposé (ensemble des attributs contenus dans le filtre temporel), l'administrateur peut associer une ou plusieurs fonctions d'agrégation. Les attributs n'étant pas associé à une fonction d'agrégation ne participent pas au filtre d'archives ; leurs évolutions ne sont pas archivées.

L'administrateur peut préciser une unité temporelle pour archiver les évolutions suivant une critère temporel de groupement afin de modérer l'archivage (*cf.* chapitre II). Il peut également

spécifier le nombre d'instants de groupement (par défaut, les groupements sont effectués pour chaque instant à l'unité temporelle choisie). Un état archivé est décrit par groupe d'instants.

3.3.4 Exemple complet d'historisation

EXEMPLE : Nous poursuivons l'exemple de l'entrepôt élaboré à la section 3.2.5. Pour les besoins des décideurs, l'administrateur veut conserver les évolutions des établissements hospitaliers ainsi que des services qui les composent. Il définit donc un environnement de la manière suivante :

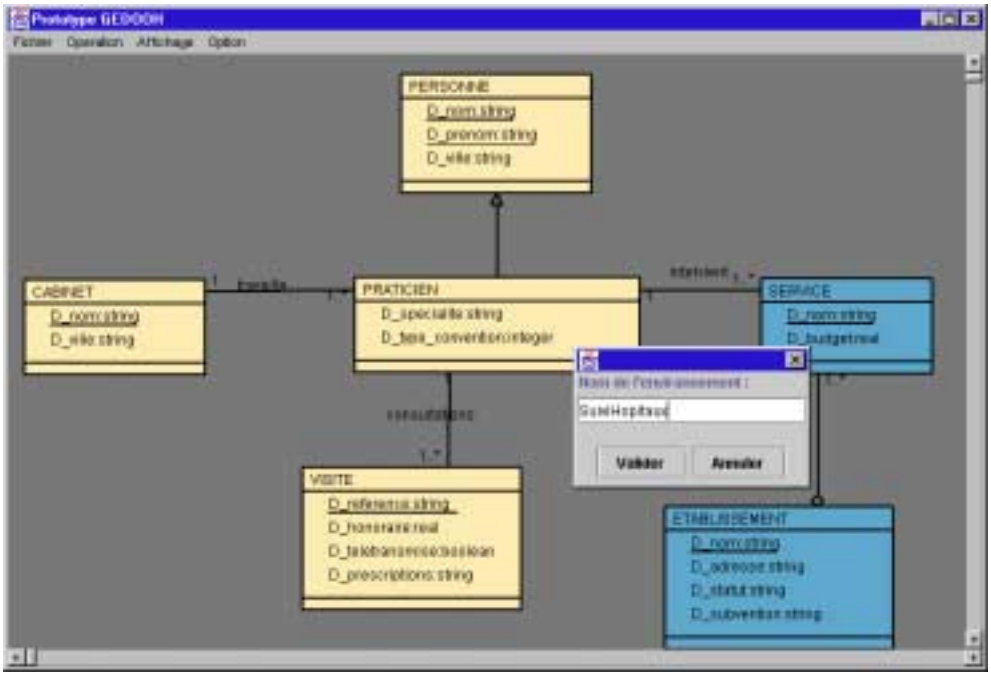


Figure 61 : Définition de l'environnement "*SuiviHopitaux*".

L'environnement ainsi défini, l'administrateur peut caractériser les filtres temporels et d'archives des classes contenues dans *SuiviHopitaux*.

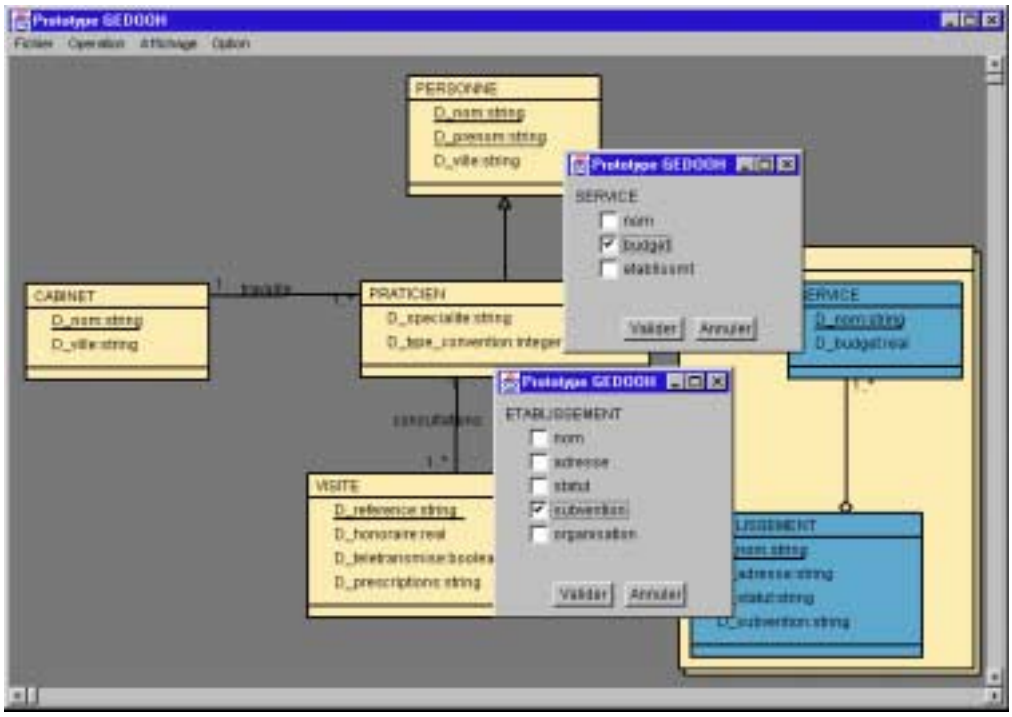


Figure 62 : Définition des filtres temporels de "*Etablissement*" et "*Service*".

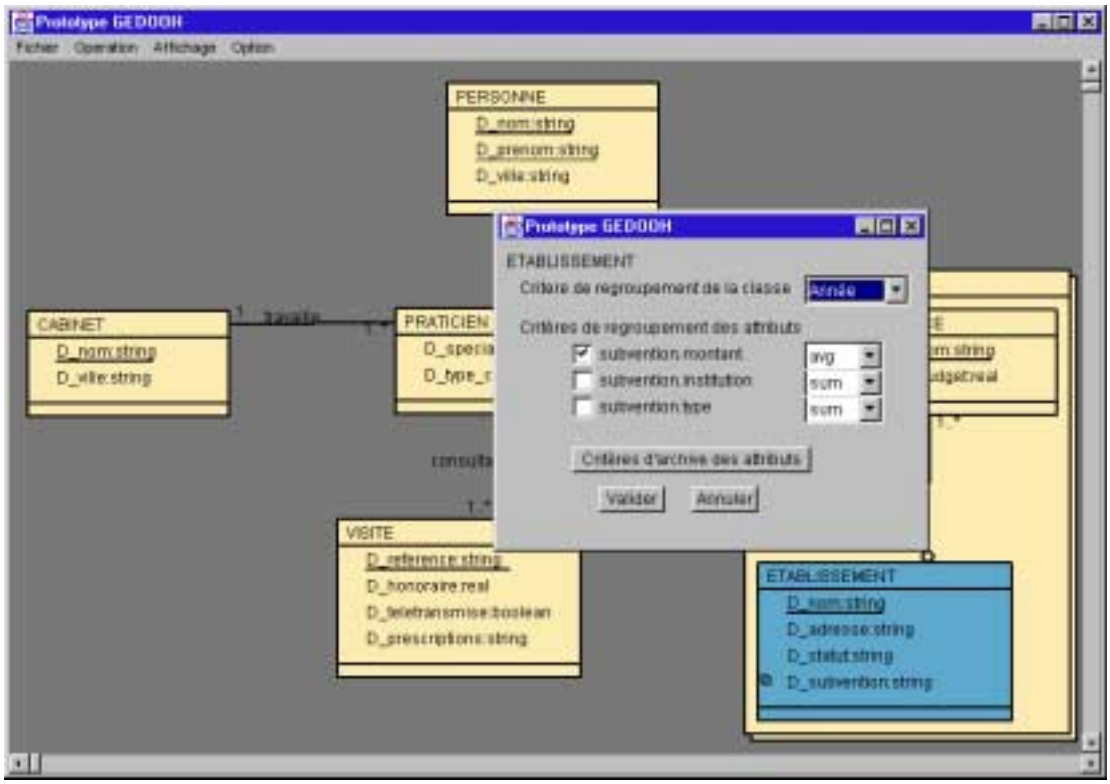


Figure 63 : Définition du filtre d'archives de "*Etablissement*".

3.4 Configuration

La dernière étape consiste à définir les configurations de l'entrepôt ainsi que des environnements. Il s'agit, pour l'administrateur de caractériser le comportement de l'entrepôt et des environnements en définissant des règles fixant des périodes de rafraîchissement, des critères d'archivage (déterminant les états passés à archiver)… Se reporter au chapitre II pour plus de détails.

L'administrateur dispose de deux possibilités.

- Il peut définir certaines règles qui sont essentielles pour le fonctionnement de l'entrepôt (période de rafraîchissement de l'entrepôt, critères d'archivage de chaque environnement). La définition de ces règles est assistée par le système.
- Il peut également définir des règles supplémentaires (périodes de rafraîchissement spécifique aux environnements). La définition de ces règles est libre et demande des compétences importantes de la part de l'administrateur.

3.4.1 Définition assistée des règles

La définition assistée de règles de configuration permet à l'administrateur de gérer les processus d'historisation et d'archivage des données de l'entrepôt ; ces règles sont nécessaires au fonctionnement de celui-ci. Parmi les contraintes que comporte la configuration des environnements, nous trouvons les critères d'archives qui permettent de sélectionner les états passés à archiver. Ces critères peuvent être fixes ou dynamiques. GEDOOH assiste l'administrateur pour définir ces critères.

3.4.1.1 Critères d'archivage fixe

Un critère d'archivage est une contrainte dans laquelle intervient le domaine temporel des états passés. L'expression temporelle est fixe si le domaine temporel des états passés qualifiés pour l'archivage est comparé à une date, c'est à dire que la valeur de cette expression ne change pas avec l'avancement du temps présent.

L'expression assistée d'une règle d'archivage fixe est réalisée au travers d'une fenêtre.

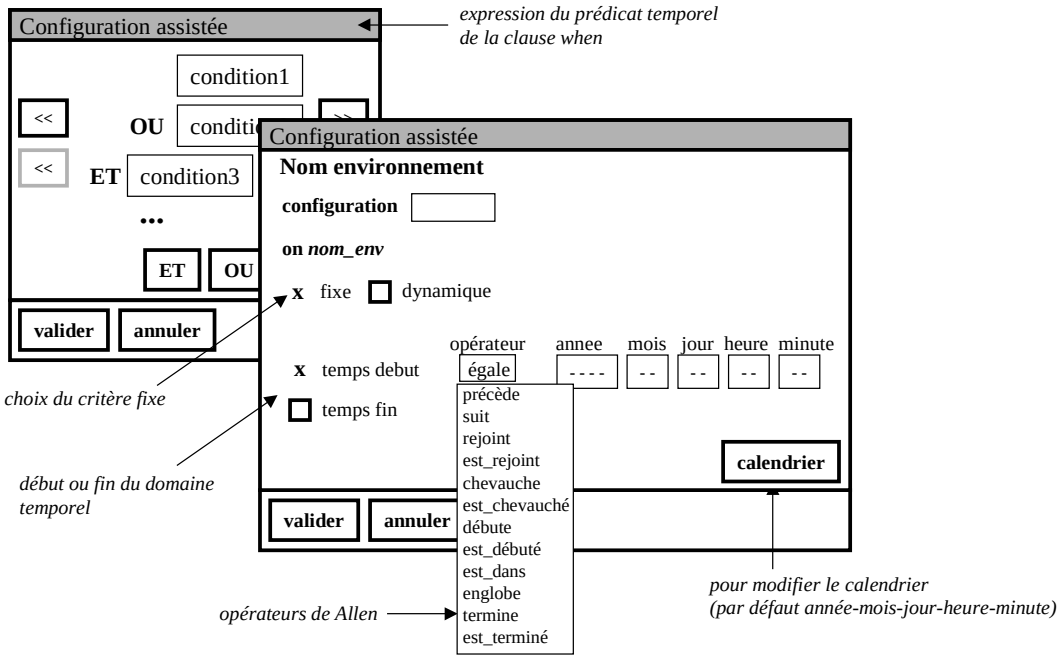


Figure 64 : Définition assistée d'un critère d'archivage fixe.

3.4.1.2 Critères d'archivage dynamique

Pour créer un critère d'archivage dynamique, l'expression temporelle est dynamique, c'est à dire que le domaine temporel des états passés à archivés est comparé à une valeur temporelle comportant une référence au temps présent (now). Ainsi, la règle de configuration est évolutive avec l'avancement du temps présent (contrairement aux critères fixes).

L'expression assistée d'une règle d'archivage dynamique est réalisée au travers de la même fenêtre que pour les critères fixes, mais en cochant le choix d'une règle dynamique.

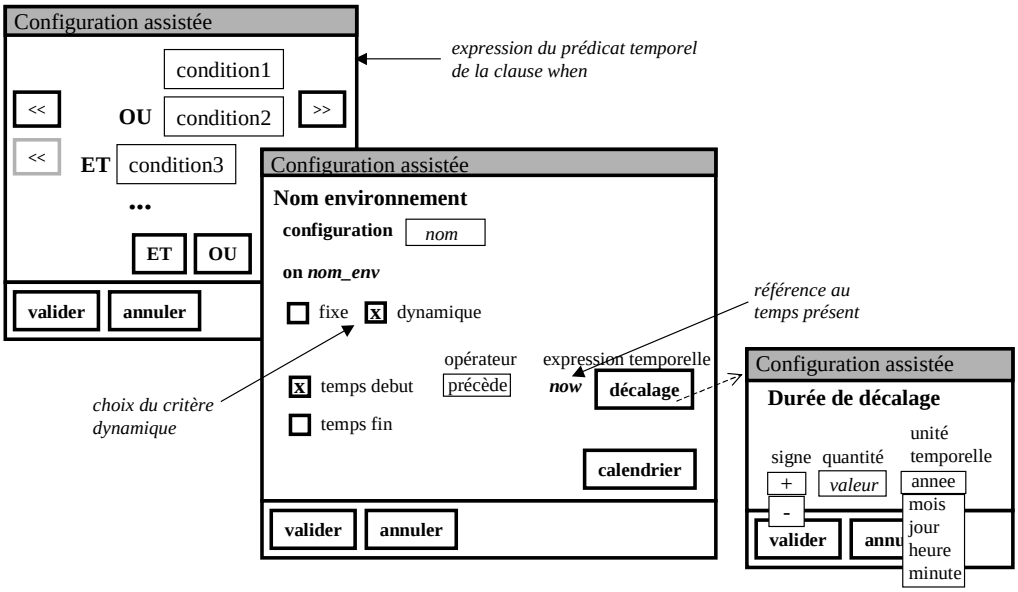


Figure 65 : Définition assistée d'un critère d'archivage dynamique.

3.4.2 Définition libre des règles

La configuration libre est réalisée au travers d'une fenêtre de saisie, dans laquelle l'administrateur définit une règle au travers du langage ECA décrit au chapitre II.

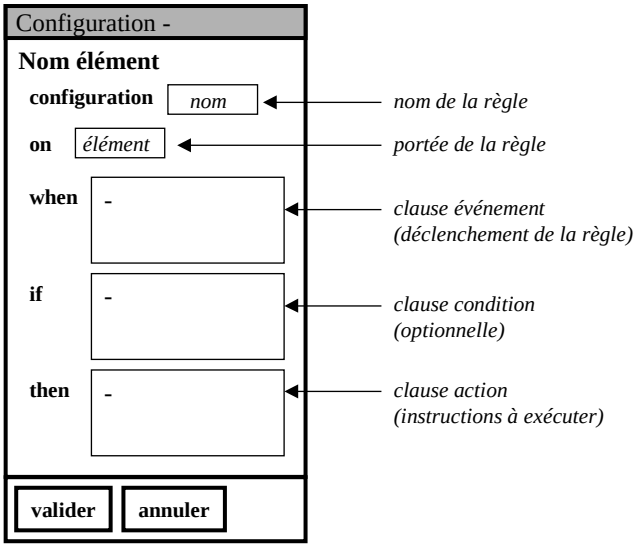


Figure 66 : Définition d'une règle de configuration.

Il s'agit d'un module qui s'adresse à un administrateur expérimenté, qui doit maîtriser le langage de définition des règles de configuration. Ceci n'est pas rédhibitoire car la configuration exige des compétences et des connaissances importantes de la part de l'administrateur.

Cette étape reste donc complexe mais essentielle car elle détermine en partie l'information stockée dans l'entrepôt ; en particulier, les règles de configuration définissent :

- les périodes avec laquelle l'entrepôt et les environnements sont rafraîchis ainsi que
- la gestion des données historisées et archivées.

3.4.3 Exemple complet de configurations

EXEMPLE : Nous poursuivons l'exemple de l'entrepôt élaboré à la section 3.3.4. L'administrateur configure l'environnement par une règle d'archivage qui caractérise les états passés à archiver : il s'agit des états passés dont la valeur est conservée depuis plus de cinq ans dans l'entrepôt. Pour cela, il utilise la configuration assistée qui facile sa tâche.

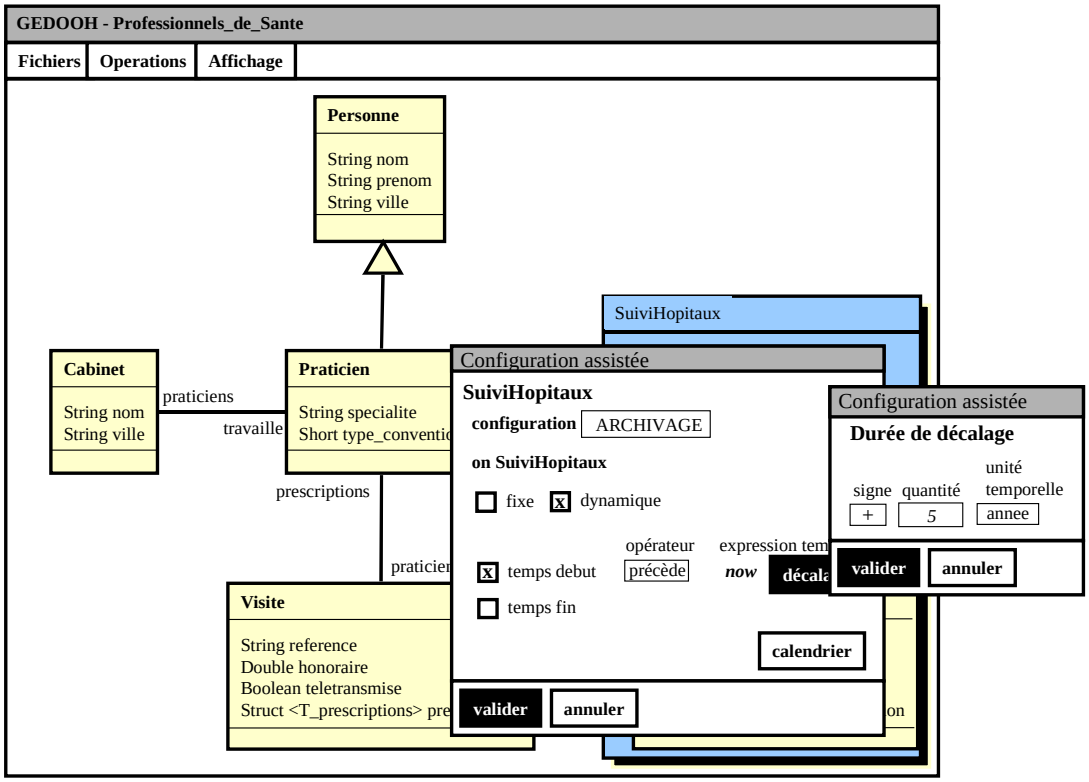


Figure 67 : Définition du filtre d'archives de "*Etablissement*".

3.5 Synthèse

Nous avons décrit une démarche d'élaboration du graphe de l'entrepôt à partir de celui d'une source globale. Cette démarche comprend trois étapes : l'extraction, l'historisation et la configuration. Chacune de ces étapes est réalisée au travers de l'interface graphique proposée dans GEDOOH qui se caractérise par un fonctionnement incrémental, interactif, uniforme et flexible.

Actuellement, l'outil GEDOOH comprend environ 6000 lignes de code Java (jdk 1.2) pour l'interface et le module de création des entrepôts. L'étape d'extraction est en cours d'implantation tandis que les deux dernières étapes de notre démarche (l'historisation et la configuration) ont été validées au travers de leur implantation.

Nous avons présenté le langage graphique supporté par GEDOOH. Ce langage est basé sur l'utilisation d'un graphe source pour élaborer celui de l'entrepôt. Notre langage permet de définir graphiquement les fonctions de construction des classes entrepôt, leurs filtres temporels et d'archives ainsi que les environnements et les règles de configuration. La configuration est réalisée soit de manière assistée, soit de manière libre (nécessitant alors pour l'administrateur des compétences plus importantes).

4 GENERATION AUTOMATIQUE D'ENTREPOTS DANS UN SGBD RELATIONNEL

Dans le cadre du projet REANIMATIC, notre partenariat avec des hôpitaux parisiens nous a amené à travailler avec un SGBD relationnel. Ce choix offre les avantages de puissance, de grande capacité de stockage et de performance lors de la manipulation des données. En outre ces systèmes relationnels (comme celui d'Oracle) existent maintenant depuis plusieurs années, ce qui permet d'assurer le suivi, la maintenance et l'évolution des systèmes.

4.1 Metadonnées relationnelles

Pour générer un entrepôt de données relationnelles, nous utilisons une métabase décrivant les caractéristiques de l'entrepôt. cette métabase comprend des données décrivant les structures de l'entrepôt.

La Figure 68 présente le métamodèle conceptuel qui sert à la description des caractéristiques de la métabase relationnelle de notre entrepôt. Ce métamodèle est décrit selon le modèle Entité/Association [Chen 1976].

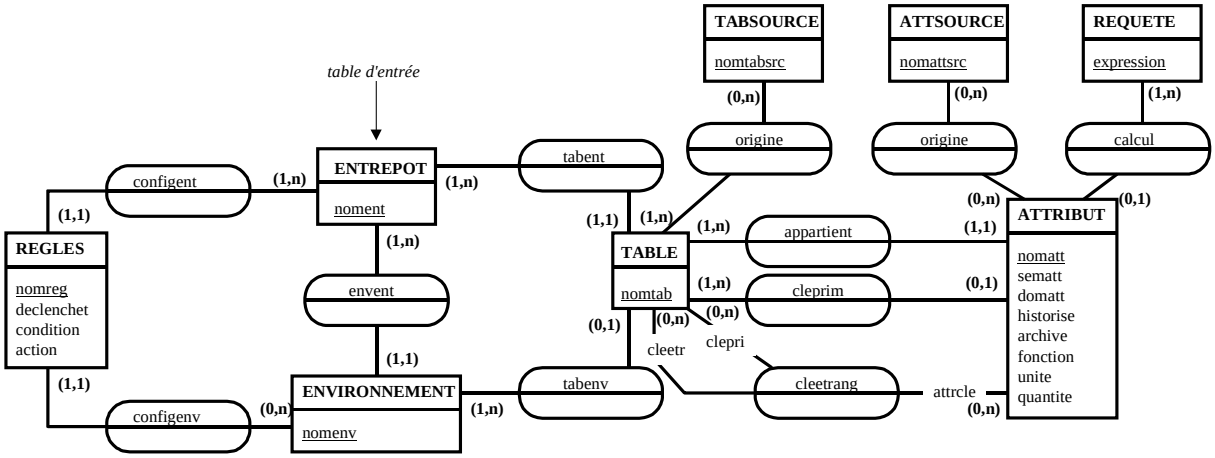


Figure 68 : Modèle conceptuel de la métabase relative au modèle relationnel d'entrepôts.

Un entrepôt de données est composé d'un ensemble de tables, d'environnements et de règles de configuration.

Une table est historisée si elle comporte au moins un attribut historisé, qui peut être éventuellement archivé en définissant une fonction d'agrégation (forte ou modérée). Chaque table comporte un ensemble d'attributs, pouvant être :

- des attributs caractérisant la table,
- des attributs clés primaires définissant la table,
- des attributs clés étrangères exprimant des contraintes d'intégrité référentielles avec d'autres tables de l'entrepôt.

Les attributs sont définis par un nom et caractérisés par son domaine et sa sémantique (puisqu'ils peuvent être dérivés, calculés ou spécifiques).

Chaque environnement comporte les tables qui suivent le même comportement temporel. Ce dernier est décrit par des règles de configuration.

Les règles de configuration sont définies par son nom. Elle se caractérise par une expression de déclenchement, parfois par une expression qui conditionne le déclenchement et par une expression décrivant l'action de la règle.

Ce métamodèle conceptuel est transformé au niveau logique en un modèle relationnel implanté par la suite dans une base relationnelle. Cette métabase contient les métadonnées permettant de générer le schéma de l'entrepôt, mais elles n'indiquent pas comment obtenir l'entrepôt à partir du graphe de l'interface GEDOOH.

4.2 Limitations relationnelles

Le module générateur de GEDOOH est chargé de l'implantation automatique de l'entrepôt dans le SGBD relationnel. Or les structures relationnelles imposent des restrictions par rapport aux possibilités offertes par notre modèle conceptuel orienté objet dédié aux entrepôts.

- Au niveau des concepts inhérents à l'objet.

  Les classes et les liens doivent être traduit en relationnel par des relations et des clés primaires et étrangères.

  Le domaine des attributs est limité aux types simples, nécessitant de transcrire les attributs multivalués en relationnel. En particulier, les attributs temporels des états des objets sont définis par le type complexe de domaine temporel qui doit être traduit.

  Les concepts de l'approche objet, plus riches, doivent être traduit avec les concepts réduits de l'approche relationnelle. Cela porte principalement sur l'héritage et les agrégations.

- Au niveau des concepts inhérents à notre modélisation.

  Les classes entrepôt intègrent des filtres temporels et d'archives (modélisation des évolutions détaillées et résumées) qui définissent les structures des états passés et archivés des objets entrepôt. Il est nécessaire de traduire les concepts d'états courants, passés et archivés par des n-uplets stockés dans des relations.

En outre, nous n'abordons pas ici les opérations (ou méthodes) associées aux classes entrepôt. Il n'est pas envisagé d'exporter automatiquement des opérations objet en relationnel. Toutefois, nous avons mené une étude concernant la réalisation de la dérivation automatique des opérations vers un SGBD objet (O2) dans [Ravat, Teste, Zurfluh 2000a].

4.3 Passage du niveau conceptuel objet au niveau logique relationnel

Cette section se propose de définir les règles de passage du niveau conceptuel objet vers le niveau logique relationnel. Nous définissons un premier ensemble de règles permettant de transformer les concepts objet standard (classe, héritage, agrégation), puis un second ensemble de règles permettant de transformer les concepts inhérents à notre modèle d'entrepôt (état passé, état archivé).

4.3.1 Transformations des concepts objet

Nous proposons six règles de passage pour transformer les concepts objet standards de classe, d'héritage et d'agrégation en relationnel.

4.3.1.1 Les classes d'objets

R1. Une classe d'objets est représentée par une relation (table) de même nom ayant pour attributs la liste des attributs de la classe.

Chaque relation résultante de la règle R1 doit posséder une clé primaire, c'est à dire un groupe d'attributs qui détermine les n-uplets de manière unique dans la relation.

4.3.1.2 Les classes d'association

R2. Une classe d'association est représentée par une relation de même nom ayant pour attributs la liste des clés des classes objets reliées et les attributs propres à la classe d'association.

Les relations résultantes des classes d'association ont pour clé primaire la concaténation de l'ensemble des clés des relations correspondantes aux classes d'objets impliquées.

Il est possible de regrouper les relations d'association binaire sans classe d'association.

R3. Une classe d'association bijective (de cardinalités minimale et maximale 1) est regroupée dans la relation correspondant à la classe d'objets de cardinalité 1. La clé de la relation correspondant à la deuxième classe d'objets devient une clé étrangère dans la relation du regroupement.

4.3.1.3 L'héritage

Les systèmes relationels ne connaissent pas le concept d'héritage. Il faut donc réaliser statiquement ce dernier lors du passage du niveau objet vers le niveau relationnel. Plusieurs solutions sont possibles [Gardarin 1999], mais la solution la plus naturelle pour traduire la généralisation des classes C_1, C_2,…, C_n vers une classe C (ou la spécialisation de C vers les classes C_1, C_2,…, C_n) est d'appliquer le règle R1 qui conduit à traduire chaque classe d'objets en une relation munie d'une clé primaire, puis d'appliquer la règle suivante.

R4. La spécialisation d'une classe C en plusieurs classes C_1, C_2,…, C_n est traduite par une répétition de la clé primaire de la relation représentant C dans chaque relation représentant C_1, C_2,…, C_n.

Cette solution conduit à une relation par classe. Pour retrouver un attribut hérité, il faut effectuer une jointure avec la relation représentant la super-classe.

4.3.1.4 L'agrégation

L'agrégation composite correspond à un groupe d'attributs imbriqués dans la classe composite. Dans le cas de bijection, tous les attributs de la classe composante sont simplement ajoutés à la relation représentant la classe composée.

R5. Une classe dépendante d'une autre par une agrégation composite monovaluée est représentée par des attributs ajoutés à la relation représentant l'objet composite.

Le cas d'une agrégation indépendante est traduit en appliquant la règle R2.

Le relationnel ne permet pas de traduire une relation composite multivaluée par une seule relation. On procède alors comme dans le cas d'une agrégation indépendante et plus généralement une association.

4.3.1.5 Les attributs multivalués

R6. Un attribut multivalué est traduit par une relation à part, ayant comme clé primaire, celle de la relation issue de la classe d'objets contenant l'attribut.

4.3.2 Transformations des concepts inhérents à notre modélisation des entrepôts

Notre modèle intègre plusieurs concepts temporels qui lui permettent de conserver l'évolution des valeurs des données sous forme détaillée ou archivée. Ces concepts ne sont pas pris en compte par les règles précédentes. Nous proposons de compléter celles-ci par de nouvelles règles chargées de traduire tous les concepts de notre modèle.

4.3.2.1 L'historisation par les états passés

La conservation détaillée des évolutions des données se fait à l'aide d'états passés, dont la structure est définie par le filtre temporel des classes (attributs historisés). Une telle classe, munie d'un filtre temporel appartient à un environnement. Elle est traduite au niveau relationnel par deux relations : une relation contient les états courants des objets de la classe et l'autre relation contient les états passés.

R7. Toute classe munie d'un filtre temporel (contenue dans un environnement) est transformée en deux relations.

- Une relation des états courants VC-R (R désigne le nom de la classe), ayant pour attributs l'ensemble des attributs de la classe et deux attributs systèmes TD (temps de début) et TF (temps de fin).
- Une relation des états passés VP-R, ayant pour attributs l'ensemble des attributs du filtre temporel et deux attributs systèmes TD (temps de début) et TF (temps de fin).

La clé primaire de la relation VC-R est obtenue de façon identique à la règle R1 concernant la transformation des classes d'objets. La clé primaire de la relation VP-R est obtenue par concaténation de la clé de la relation VC-R et de l'attribut système TD.

Dues aux limites de l'approche relationnelle, les valeurs sont datées par des intervalles, et non pas des domaines temporels. Néanmoins, il est possible de simuler le domaine temporel de manière très simple en appliquant le règle R6 et en définissant une vue sur chaque relation, regroupant les intervalles pour lesquels les valeurs des autres attributs sont identiques.

Nous rappelons que notre modèle d'entrepôt supporte trois granularité d'historisation (attribut, classe et ensemble). Parmi ces granularités, la granularité ensemble permet notamment de conserver les évolutions détaillées des associations reliant les classes d'objets contenues dans l'environnement. La traduction de ces associations est réalisée par l'application de la règle R2 étendue par la relation des états passés VP-R définie selon R7.

4.3.2.2 L'archivage par les états archivés

Une caractéristique essentielle de notre modélisation des entrepôts est la possibilité d'archiver les états passés afin de résumer les évolutions détaillées et de ne garder que l'information pertinente pour le système d'aide à la décision. L'archivage est réalisé au travers des états archivés, définis par les filtres d'archivage des classes entrepôt.

La traduction d'une classe munie d'un filtre d'archivage (et donc nécessairement d'un filtre temporel) donne lieu à trois relations comportant respectivement l'ensemble des états courants, l'ensemble des états passés et l'ensemble des états archivés.

R8. Toute classe munie d'un filtre d'archivage est transformée en trois relations.

- La relation des états courants VC-R (obtenue suivant R7).
- La relation des états passés VP-R (obtenue suivant R7).
- La relation des états archivés VA-R (R désigne le nom de la classe), ayant pour attributs l'ensemble des attributs du filtre d'archivage et deux attributs systèmes TD (temps de début) et TF (temps de fin).

La clé primaire de VA-R est obtenue par concaténation de la clé de la relation VC-R et de l'attribut système TD.

Les valeurs des attributs de la relation VA-R sont obtenues par application des fonctions d'agrégation (définies dans le filtre d'archivage) à un sous-ensemble des n-uplets de la relation VP-R (les n-uplets de VP-R archivés sont supprimés). Le sous-ensemble des n-uplets à archiver est calculé par l'intermédiaire des règles de configuration relatives au critère d'archivage ; elles sont traduites par des requêtes de sélection applicables sur VP-R.

4.4 Synthèse

Cette section a présenté une solution pour implanter en relationnel des entrepôts de données, décrits au niveau conceptuel avec notre modèle de représentation. Pour réaliser cette génération, nous avons défini une métabase décrivant l'entrepôt et un ensemble de règles de passage du niveau conceptuel vers le niveau logique. Ces règles concernent essentiellement la traduction en relationnel des concepts objets standards et des concepts spécifiques à notre modèle d'entrepôt.

5 CONCLUSION

Ce chapitre concerne la réalisation du prototype GEDOOH, développé au sein de notre équipe SIG, et dédié à l'aide aux administrateurs pour la conception d'entrepôts de données complexes, temporelles et archivées. GEDOOH comprend :

- une interface graphique qui visualise la source globale et l'entrepôt de données par des graphes et qui permet à l'administrateur de concevoir de manière incrémentale le graphe caractérisant l'entrepôt,

- un générateur automatique de l'entrepôt qui s'appuie sur le graphe de l'entrepôt pour produire des scripts de création, d'initialisation et de rafraîchissement de l'entrepôt dans un SGBD hôte.

Nous avons proposé une démarche d'élaboration du graphe de l'entrepôt à partir de celui de la source globale. Cette démarche, réalisée au travers de l'interface graphique de GEDOOH (elle se caractérise par un fonctionnement incrémental, interactif, uniforme, flexible et ergonomique) comprend trois étapes :

- l'extraction pour construire les structures des classes entrepôt à partir de celles de la source et la réorganisation pour adapter les structures et la hiérarchie d'héritage des classes entrepôt,
- l'historisation pour définir les environnements, les filtres temporels et d'archives et
- la configuration pour définir le comportement de l'entrepôt et des environnements.

Les étapes d'historisation et de configuration sont validées (leur implantation a été réalisée) tandis que nous somme en cours d'implantation de l'étape d'extraction.

Nous avons présenté le langage graphique supporté par GEDOOH qui assiste l'administrateur à chaque étape de la démarche de conception. Ce langage est basé sur l'utilisation du graphe source pour élaborer celui de l'entrepôt. Notre langage permet de définir graphiquement les fonctions de construction des classes entrepôt, leurs filtres temporels et d'archives ainsi que les environnements et les règles de configuration. La configuration est réalisée soit de manière assistée, soit de manière libre (nécessitant alors pour l'administrateur des compétences plus importantes).

Enfin, nous avons proposé une solution permettant de traduire les entrepôts objet (au niveau conceptuel) en relationnel (au niveau logique). Cette étude est notamment motivée par notre partenariat avec des hôpitaux parisiens dans le cadre du projet REANIMATIC. Nous avons défini un ensemble de règles réalisant la traduction en relationnel des concepts objets standards et des concepts spécifiques à notre modèle d'entrepôt.

CONCLUSION

Bilan

Nos travaux traitent de la modélisation conceptuelle et de la manipulation des données dans les systèmes d'aide à la prise de décision. Ces systèmes contiennent l'information utile aux décideurs et conservent l'historique de cette information pour supporter efficacement les analyses et les processus de prise de décision. Le cadre applicatif de nos travaux est un contexte médical qui face à la profusion d'informations hétérogènes, complexes et évolutives nécessite la conception et le développement de systèmes décisionnels adaptés.

Notre approche repose sur une distinction de deux espaces de stockage : l'**entrepôt** et les **magasins** de données. L'entrepôt de données contient toute l'information utile à la prise de décision tandis qu'un magasin de données constitue un extrait de l'entrepôt pour répondre à un besoin d'analyse particulier. A partir de cette séparation, nous proposons des modèles conceptuels de représentation et des langages de manipulation des données dédiés à chacun de ces espaces de stockage.

- **Modélisation de l'entrepôt de données**. Le modèle conceptuel que nous avons défini, apporte des solutions nouvelles répondant aux exigences des entrepôts. Il intègre des objets complexes constitués d'états courants, passés et archivés. L'intérêt de notre approche est de conserver les données actuelles mais également leurs évolutions sous forme détaillée ou résumée. Nous avons étendu le concept standard de classe pour prendre en compte les caractéristiques temporelles des objets entrepôt : une classe entrepôt est donc définie par des filtres temporels et d'archives ainsi que par une fonction de construction modélisant le processus d'extraction à partir duquel la classe entrepôt est élaborée. Ce processus d'extraction tient compte de l'aspect statique mais également dynamique des données. Enfin, nous avons introduit le concept d'environnement qui permet de définir simplement différentes parties temporelles cohérentes, de taille adaptée aux besoins décisionnels et dont le comportement temporel est caractérisé par des règles de configuration. Ce nouveau concept permet d'unifier les niveaux d'historisation (attribut, classe ou ensemble).
- **Manipulation des données de l'entrepôt**. La manipulation des informations contenues dans l'entrepôt est réalisée au travers d'une algèbre regroupant un ensemble d'opérateurs qui permettent d'interroger de façon uniforme les objets et leurs états. Notre algèbre reprend les principaux opérateurs des algèbres pour objets et leurs extensions temporelles. Elle introduit également des opérateurs spécifiques à notre modélisation pour manipuler les différentes catégories d'états qui composent les objets entrepôt et pour transformer les données en séries temporelles d'états afin d'offrir différentes perspectives sur les informations et leurs évolutions de manière à faciliter la compréhension et l'analyse du contenu de l'entrepôt.
- **Modélisation des magasins de données**. Le modèle conceptuel que nous avons défini pour les magasins multidimensionnels constitue une généralisation des modèles multidimensionnels habituellement proposés (constellation de faits et dimensions munies de hiérarchies multiples). L'intérêt de notre modèle réside dans sa capacité à se détacher des choix logiques (ROLAP, OOLAP, MOLAP) et physiques. En outre, une contribution importante de notre étude concerne la spécification d'une démarche pour la transformation des données complexes, évolutives et archivées de l'entrepôt dans un magasin de données multidimensionnelles.

- **Manipulation des magasins**. La manipulation des données contenues dans les magasins est réalisée au travers d'une algèbre comportant un ensemble d'opérateurs multidimensionnels. Nous avons défini des opérations issues de l'algèbre relationnelle pour appliquer des opérations de manipulation habituelles sur le schéma multidimensionnel d'un magasin. En outre, nous avons décrit différents opérateurs permettant de transformer la structure du schéma et la granularité d'observation des données dans le magasin. Notre algèbre intègre, non seulement, l'essentiel des opérations actuellement proposées, mais également, elle étend leur fonctionnalité à notre modèle multidimensionnel en constellation.

De plus, nous avons spécifié et réalisé un outil d'aide à la conception qui produit automatiquement des entrepôts de données complexes et évolutives dans un SGBD hôte. L'intérêt de cet outil est que l'administrateur élabore l'entrepôt de données à partir d'une source globale intégrée de manière graphique et incrémentale. Notre outil s'appuie sur des graphes et intègre un langage graphique permettant d'élaborer simplement l'entrepôt en trois étapes : l'extraction, l'historisation et la configuration. L'extraction consiste à définir graphiquement des fonctions de construction (extraction et réorganisation). L'historisation permet de définir les environnements et de préciser les filtres des classes entrepôt. La configuration revient à caractériser le comportement temporel des environnements.

Perspectives

Les perspectives que nous envisageons de conduire sont les suivantes :

- **Extensions du modèle de l'entrepôt**. Dans ce mémoire de thèse nous avons abordé l'évolution de valeurs des données contenues dans l'entrepôt. Il est nécessaire de prévoir l'extension de ce modèle pour prendre en compte les évolutions de schéma (modifications du schéma des sources, suppression ou ajout de sources…). Une voie possible est l'intégration de versions de classes permettant de décrire l'évolution des schémas des classes [Teste 1996] [Hubert 1997] [Le Parc 1997].
  En outre, nous envisageons de poursuivre une proposition initiée dans [Ravat, Teste 2000b] pour alimenter notre système décisionnel par des extractions réalisées sur le Web. La prise en compte des documents non structurés ou semi-structurés du Web nécessite d'étendre notre modèle d'entrepôt.

- **Etude de la méta-modélisation des systèmes décisionnels**. Nous voulons mener une étude globale concernant la méta-modélisation des systèmes décisionnels qui couvre l'ensemble des composantes de notre architecture : l'intégration des sources, la construction de l'entrepôt, la réorganisation dans les magasins et la manipulation multidimensionnelle des données. Cette méta-modélisation doit permettre l'intégration des contraintes de la source globale, de l'entrepôt et des magasins ; ces contraintes portent aussi bien sur les modèles que sur la démarche de conception du système décisionnel.

- **Proposition d'une méthode pour la conception des systèmes décisionnels**. A l'heure actuelle, aucune méthode n'est proposée pour aider les entreprises à construire un système décisionnel. Nous voulons proposer une solution pour concevoir un système décisionnel en nous basant sur une méthode complète (modèles, formalismes, démarche et outils). Cette méthode portera sur une extension du travail présenté dans ce mémoire de thèse afin d'offrir un cadre précis et complet pour concevoir des systèmes décisionnels comportant des données complexes et évolutives. La méthode doit offrir les mécanismes nécessaires à chacune des trois grandes phases de la conception : l'intégration des sources, la construction de l'entrepôt et la réorganisation dans les magasins.

REFERENCES BIBLIOGRAPHIQUES

A

[Abiteboul, Bonner 1991] Abiteboul S., Bonner A.J., "*Object and Views*", Proceedings of the ACM SIGMOD International Conference on Management of Data, Denver (Colorado, USA), May 29-31, 1991.

[Agrawal, et al 1997] Agrawal R., Gupta A., Sarawagi A., "*Modeling Multidimensional Databases*", ICDE'97.

[Amghar, Flory 1994] Amghar Y., Flory A., "*Contraintes d'intégrité dans les bases d'objets*", Revue Ingénierie des Systèmes d'Information (ISI), Vol.2, N°3, 1994.

[Andonoff, et al 1995] Andonoff E., Hubert G., Le Parc A., Zurfluh G., "*Modelling inheritance, composition and relationship links between objects, object versions and class versions*", Proceedings of International Conference on Advanced Information Systems Engineering - CAISE'95, Jyväskylä (Finland) June 1995.

[Allen 1983] Allen J., "*Maintaining Knowledge About Temporal Intervals*", Communications de l'ACM, 26(11), pp.832-843, 1983.

[Atkinson, et al 1989] Atkinson M.P., Bancilhon F., DeWitt D.J., Dittrich K.R., Maier D., Zdonik S.B., "*The Object-Oriented Database System Manifesto*". In Proceedings of the International Conference on Deductive and Object-Oriented Databases - DOOD'89, Kyoto (Japan), December 1989.

B

[Baralis, et al 1997] Baralis E., Paraboschi S., Teniente E., "*Materialized view selection in a multidimensional database*", Proc. VLDB '97.

[Beech, Mahbod 1988] Beech D., Mahbod B., "*Generalized version control in an object-oriented database*", Proceedings of the 4th International Conference on Data Engineering, Los Angeles (USA), 1988.

[Bellahsene 1998] Bellahsene Z., "*View Adaptation in Data Warehousing Systems*", Proceedings of the 9th International Conference on Database and Expert Systems - DEXA'98.

[Bellahsene 1996] Bellahsene Z., "*View Mechanism for Schema Evolution*", Proceedings of the 14th British National Conference on Databases - BNCOD'96, Edinburgh, UK, July 3-5, 1996.

[Bestougeff, Ligozat 1989] Bestougeff H., Ligozat G., "*Outils logiques pour le traitement du temps, de la linguistique à l'intelligence artificielle*", Chapitre 2.4, Systèmes d'Allen et intervalles généralisés, pp65-82, ed. Masson, ISBN 2-225-81 632-8, 1989.

[Bertino, et al 1996] Bertino E., Ferrari E., Guerrini G., "*A Formal Temporal Object-Oriented Data Model*", Proceedings of the 5th International Conference on Extending Database Technology - EDBT'96, Avignon (France), 1996.

[Bertino, et al 1998a] Bertino E., Ferrari E., Guerrini G., Merlo I., "*Extending the ODMG Object Model with Time*", Proccedings of the 12th European Conference on Object-Oriented Programming - ECOOP'98, Brussels (Belgium), July 20-24, 1998.

[Bertino, et al. 1998b] Bertino E., Ferrari E., Guerrini G., Merlo I., "*Extending the ODMG Object Model with Composite Objects*", Proceedings of the 1998 ACM SIGPLAN Conference

on Object-Oriented Programming Systems, Languages & Applications - OOPSLA'98, Vancouver (British Columbia, Canada), October 18-22 1998.

[Björnersted, Hultén 1989] Björnersted A., Hultén C., "*Version control in object-oriented architecture*", Object-oriented concepts, databases and applications, ed. W. Kim (F. Lochovsky) - Addison-Wesley publishing compagny, 1989.

[Bret, et al 2000] Bret F., Soule-Dupuy C., Zurfluh G., "*Outils méthodologiques pour la conception de bases de données décisionnelles orientées objet*", LMO 2000, St Hilaire (Canada), Janvier 2000.

[Bret, Teste 1999] Bret F., Teste O., "*Construction Graphique d'Entrepôts et de Magasins de Données*", Actes du XVIIème Congrès INFormatique des ORganisations et Systèmes d'Information et de Décision - INFORSID'99, *ed*. INFORSID - ISBN 2-906855-15-4, p165-184, 2-4 Juin 1999, La Garde (Var, France).

[Bukhres, Elmagarmid 1993] Bukhres O.A., Elmagarmid A.K., "*Object-Oriented Multidatabase Systems - A solution for Advanced Applications*", Prentice Hall, ISBN 0-13-103813-2, 1993.

[Buzydlowski, et al 1998] Buzydlowski J.W., Song I.Y., Hassell L., "*A Framework for Object-Oriented On-Line Analytic Processing*", DOLAP'98, Bethesda (Maryland, USA), 7 November 1998.

C

[Cabibbo, Torlone 1997] Cabibbo L., Torlone R., "*Querying Multidimensional Databases*", Proceedings of the 6th DBPL Workshop, pp 253-269, 1997.

[Cabibbo, Torlone 1998] Cabibbo L., Torlone R.,"*From a Procedural to a Visual Query Language for OLAP*", IEEE International Conference on Scientific and Statistical Database Management, SSDBM'98, 1998.

[Canavaggio 1997] Canavaggio J-F., "*TEMPOS : un modèle d'historiques pour un SGBD temporel*", thèse de l'Université Joseph Fourier - Grenoble 1, 22 novembre 1997.

[Canillac 1991] Canillac M., "*Un modèle et une interface hypertexte pour les bases de données orientées objet*", Thèse de l'Université Paul Sabatier - Toulouse III, 11 juillet 1991.

[Cattel 1995] Cattel R.G.G, "*ODMG-93 Le Standard des bases de données objet*", Thomson publishing, ISBN 2-84180-006-7, 1995.

[Cauvet, Semmak 1994] Cauvet C., Semmak F., "*Abstraction Forms in Object-Oriented Conceptual Modeling: Localization, Aggregation and Generalization Extensions*", Proceedings of the 6th Advanced Information Systems Engineering - CAiSE'94, Utrecht (The Netherlands), June 6-10, 1994.

[Cellary, Jomier 1990] Cellary W., Jomier G., "*Consistancy of Versions in Object-Oriented Databases*", Proceedings of the 16th International Conference on Very Large Databases - VLDB'90, Brisbane (Australia), September 1990.

[Chaudhuri, Dayal 1997] Chaudhuri S., Dayal U., "*An Overview of Data Warehousing and OLAP Technology*", ACM SIGMOD Record, 26(1), 1997.

[Chawathe, et al 1994] Chawathe S., Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J., Widom J., "*The TSIMMIS Project: Integration of Heterogeneous Information Sources*", In Proceedings of IPSJ Conference, pp. 7-18, Tokyo, Japan, October 1994.

[Chen 1976] Chen P.P., "*The Entity-Relationship Model - Toward a Unified Views of Data*", ACM Transactions on Database Systems - ACM TODS, Vol 1,pp 9-36, Mars 1976.

[Chou, Kim 1986] Chou H.T., Kim W., "*A unifying framework for version control in a CAD environment*", Proceedings of the 12th International Conference on Very Large Databases - VLDB'86, Kyoto (Japan), August 1986.

[Chrisment, et al 1999] Chrisment C., Pujolle G., Zurfluh G, "*Langages de bases de données : SQL et les évolutions vers l'objet*", traité informatique des Techniques de l'Ingénieur - TI, 1999.

[Codd 1993] Codd E.F., "*Providing OLAP (on-line analytical processing) to user-analysts : an IT mandate*", Technical Report, E.F. Codd and Associates, 1993.

[Crimmins, et al 1998] Crimmins F., Dkaki T., Mothe J., Smeaton A. F., "*TétraFusion: Information Discovery on the Internet*", IEEE Intelligent Systems & their applications, Vol 14, N 4, pp 55-62, IEEE Computer Society, Juillet-Août 1999.

[Ctisud 1998] Convention de stage UPS (Université Paul Sabatier) - CTI-Sud (Centre de Traitement Informatique de l'Assurance Maladie de Midi-Pyrénées et Languedoc Roussillon), Septembre 1998/Août 1999.

[Cui, Widom 2000] Cui Y., Widom J., "*Practical Lineage Tracing in Data Warehouses*" In Proceedings of the Sixteenth International Conference on Data Engineering, San Diego (California, USA), February 2000.

D

[Dayal, et al. 1988] Dayal U., Blaustein B.T., Buchmann A.P., Chakravarthy U.S., Hsu M., Ledin R., McCarthy D.R., Rosenthal A., Sarin S.K., Carey M.J., Livny M., Jauhari R., "*The HiPAC Project: Combining Active Databases and Timing Constraints*", ACM SIGMOD Record, 17(1), Chicago (Illinois, USA), 1988.

[Dayal, Wuu 1992] Dayal U., Wuu G.T.J., "*A Uniform Approach to Processing Temporal Queries*", Proceeding of International Conference on Very Large Database - VLDB'92, Vancouver (Canada), 1992.

[Delobel, Adiba 1982] Delobel C., Adiba M., "*Bases de données et systèmes relationnels*", Dunod informatique, ed. Bordas, ISBN 2-04-011628-1, Paris (France), 1982.

E

[Elmashri, Wuu 1990] ElmashriR., Wuu G.T.J., "*A Temporal Model and Query Language for EER Databases*", Proccedings of the 6th International Conference on Data Engineering, February 1990.

F

[Fayyad, et al 1996] Fayyad U.M., Piatetsky-Shapiro G., Smyth P., Uthurusamy R., "*Advances in Knowledge Discovery and Data Mining*", AAAI Press, ISBN 0-262-56097-6, 1996.

G

[Gançarski 1994] Gançarski S., "*Versions et Bases de Données : modèle formel, supports de langage et d'interface utilisateur*", Thèse de l'Université Paris XI Orsay, 1994.

[Garcia-Molina, et al 1995] Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J., Widom J., "*Integrating and Accessing Heterogeneous Information Sources in TSIMMIS*", In Proceedings of the AAAI Symposium on Information Gathering, pp. 61-64, Stanford (California, USA), March 1995.

[Garcia-Molina, et al 1998] Garcia-Molina H., Labio W. J., Yang J., "*Expiring Data in a Warehouse*" In Proceedings of the 24th VLDB Conference - VLDB'98, New York (USA), August, 1998.

[Gardarin 1999] Gardarin G., "*Bases de Données : objet et relationnel*", Ed Eyrolles, ISBN 2-212-09060-9, 1999.

[Gatziu, et al 1999] Gatziu S., Jeusfeld M.A., Staudt M., Vassiliou Y., "*Design and Management of Data Warehouses*", Report on the DMDW'99, ACM SIGMOD Record, 28(4), Dec. 1999.

[Golfarelli, et al 1998] Golfarelli M., Maio D., Rizzi S., "*Conceptual Design of Data Warehouses from E/R Schemes*", In Proceedings of the 31st Hawaii International Conference on System Sciences, Kona (Hawaii, USA), 1998.

[Golfarelli, Rizzi 1999] Golfarelli M., Rizzi S., "*Designing the data warehouse: key steps and crucial issues*", Journal of Computer Science and Information Management, vol. 2, n. 3, 1999.

[Goralwalla, Özsu 1993] Goralwalla I., Özsu M.T., "*Temporal Extensions to a Uniform Bahevioral Object Model*", Proceedings of the 12th International Conference on Entity-Relationship Approach - ER'93, Dallas (Texas, USA), December 1993.

[Gupta 1997] Gupta H., "*Selection of Views to Materialize in a Data Warehouse*", In Proceedings of the International Conference on Database Theory, Athens, Greece, January 1997.

[Gupta, Mumick 1995] Gupta A., Mumick I.S., "*Maintenance of Materialized Views: Problems, Techniques, and Applications*", IEEE Data Engineering Bulletin, 1995.

[Gupta, Mumick 1999] Gupta H., Mumick I.S., "*Selection of Views to Materialize Under a Maintenace-Time Constraint*", In Proceedings of the International Conference on Database Theory - ICDT'99, Jerusalam (Israel), January 1999.

[Gyssen, Lakshmanan 1997] Gyssen M., Lakshmanan L.V.S., "*A Foundation for Multi-Dimensional Databases*", In Proceedings of 23rd International Conference on Very Large Data Bases - VLDB'97, Athens (Greece), August 25-29 1997.

H

[Hammer, et al 1995] Hammer J., Garcia-Molina H., Widom J., Labio W. J., Zhuge Y., "*The Stanford Data Warehousing Project*", IEEE Data Engineering Bulletin, June 1995.

[Harinarayan, et al 1996] Harinarayan V., Rajaraman A., Ullman J., "*Implementing Data Cubes Efficiently*", In Proceedings of ACM SIGMOD Conference, Montreal, Canada, June 1996.

[Hubert 1997] Hubert G., "*Les versions dans les bases de données orientées objet : modélisation et manipulation*", Thèse de l'Université Paul Sabatier - Toulouse III, 9 Janvier 1997, Toulouse (France).

[Hull, Zhou 1996] Hull R., Zhou G., "*A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches*", Proc. ACM SIGMOD '96.

[Hurtado, et al 1999] Hurtado C., Mendelzon A., Vaisman A., "*Maintaining Data Cubes under Dimension Updates*", Proc ICDE '99, San Diego (California, USA), March 1999.

[Huyn 1996] Huyn N., "*Efficient View Self-Maintenance*", In Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications, Montreal (Canada), June 7 1996.

[Huyn 1997] Huyn N., "*Multiple-View Self-Maintenance in Data Warehousing Environments*", In Proceedings of 23rd International Conference on Very Large Data Bases - VLDB'97, Athens (Greece), August 25-29 1997.

I

[Inmon 1994] Inmon W.H., "*Building the Data Warehouse*", John Wiley&Sons, ISBN 0471-14161-5, 1994.

J

[Jarke, et al 1998] Jarke M., Jeusfeld M.A., Quix C., Vassiliadis P., "*Architecture and quality in data warehouses*", Proceedings of the 10th Conference on Advanced Information Systems Engineering (CAiSE '98), Pisa, Italy, June, 8-12, 1998.

[Jeusfeld, et al 1998] Jeusfeld M.A., Quix C., Jarke M., "*Design and Analysis of Quality Information for Data Warehouses*", Proceedings of the 17th International Conference on Conceptual Modeling - ER'98, Singapore, Nov 16-19, 1998.

K

[Käfer, Schöning 1992a] Käfer W., Schöning H., "*Realising a Temporal Complex-Object Data Model*", Proceedings of the 8th International Conference on Management of Data - ACM SIGMOD, San Diego (Californy, USA), June 1992.

[Käfer, Schöning 1992b] Käfer W., Schöning H., "*Mapping a version model to a complex object data model*", Proceedings of the 8th International Conference on Data Engineering, Tempe (USA), February 1992.

[Kakoudakis, Theodoulidis 1996] Kakoudakis I., Theodoulidis B., "*The TAU Temporal Object Model*", Technical Report TR-96-4, TimeLab, University of Manchester (UMIST), 1996.

[Kedad, Métais 1999] Kedad Z., Métais E., "*Dealing with Semantic Heterogeneity During Data Integration*", Proceeding of the18th International Conference on Conceptual Modeling - ER'99, Paris (France), November, 15-18, 1999.

[Kimball 1996] Kimball R., "*The data warehouse toolkit*", John Wiley and Sons, 1996.

[Kotidis, Roussopoulos 1999] Kotidis Y., Roussopoulos N., "*DynaMat: A Dynamic View Management System for Data Warehouses*", Proc. ACM/SIGMOD '99, Philadephia (Pennsylvania, USA), December 1999.

[Kouramajian, Gertz 1995] Kouramajian V., Gertz M., "*A Graphical Query Language for Temporal Databases*", Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship - ER'95, Australia, December 1995.

[Kuno, Rundensteiner 1996] Kuno H.A., Rundensteiner E.A., "*The MultiView OODB View System: Design and Implementation*", Journal of Theory and Practice of Object Systems (TAPOS), Special Issue on Subjectivity in Object-Oriented Systems, Vol. 2, No. 3., Ed. Harold Ossher and William Harrison, John Wiley New York, 1996 (extended version of University of Michigan Tech. Rep. CSE-TR-246-95, May 1995).

L

[Labio, et al 1997] Labio W. J., Quass D., Adelberg B., "*Physical Database Design for Data Warehousing*", In Proceedings of the International Conference on Data Engineering, Binghamton, UK, April, 1997.

[Labio, et al 1999] Labio W. J., Yerneni R., Garcia-Molina H., "*Shrinking the Warehouse Update Window*", In Proceedings of the ACM SIGMOD Conference, Phildelphia (USA), May 1999.

[Labio, Garcia-Molina 1996] Labio W. J., Garcia-Molina H., "*Efficient Snapshot Differential Algorithms for Data Warehousing*", In Proceedings of VLDB Conference, Bombay, India, September 1996, pp. 63-74.

[Lacroix, et al 1998] Lacroix Z., Delobel C., Brèche P., "*Object Views*", Networkink and Information System Journal, Vol. 1, N°2-3 (Databases and Information Systems), 1998.

[Ladkin 1987] Ladkin P.B., "*The Logic of Time Representation*", Thesis of the University of California at Berkley, November 1987.

[Lapujade, Ravat 1997] Lapujade A., Ravat F., "*Conception de systèmes multimédia répartie : application au milieu hospitalier*", Actes du XVième Congrès INFORSID, pp163-184, Toulouse (France), 10-13 juin 1997.

[Lehner 1998] Lehner W., "*Modeling Large Scale OLAP Scenarios*", 6th International Conference on Extending Database Technology - EDBT'98, Valencia (Spain), March, 1998.

[Lehner, et al 1998] Lehner W., Albrecht J., Wedekind H., "*Multidimensional Normal Forms*", In Proceedings of the 10th International Conference on Scientific and Statistical Data Management - SSDBM'98, Capri (Italy), July 1-3, 1998.

[Le Parc 1997] Le Parc A., "*Une algèbre et un langage graphique pour les bases de données objet intégrant le concept de version*", Thèse de l'Université Paul Sabatier - Toulouse III, Décembre 1997.

[Lerner, Habermann 1990] Lerner B.S., Habermann A.N., "*Beyond Schema Evolution to Database Reorganization*", Proceedings of Conference on Object-Oriented Programming : Systems, Languages and Applications - ECOOP'90, 1990.

[Li, Wang 1996] Li C., Wang X. S., "*A Data Model for Supporting On-Line Analytical Processing*", In Proceedings of the 5th International Conference on Information and Knowledge Management - ACM CIKM'96, November 1996.

M-N

[Maier 1983] Maier D., "*The Theory of Relational Databases*", *ed*. Computer Science Press, 1983.

[Marcel 1998] Marcel P. "*Manipulation de Données Multidimensionnelles et Langages de Règles*", Thèse de Doctorat de l'Institut des Sciences Appliquées de Lyon, 1998.

[Mendelzon, Vaisman 2000] Mendelzon A.O., Vaisman A.A., "*Temporal Queries in OLAP*", Proceedings of 26th International Conference on Very Large Data Bases - VLDB 2000, Cairo (Egypt), September 10-14, 2000.

[Monk, Sommerville 1993] Monk S., Sommerville I., "*Schema evolution in object-oriented databases using class versioning*", ACM SIGMOD record, vol 22, n°3, 1993.

[Muller 2000] Muller P-A., "*Modélisation objet avec UML*", *ed*. Eyrolles, ISBN 2212091222, Mars 2000.

[Mumick, et al 1997] Mumick I., Quass D., Mumick B., "*Maintenance of Data Cubes and Summary Tables in a Warehouse*", In Proceedings of the ACM SIGMOD Conference, Tuscon (Arizona, USA), May, 1997.

O

[O2tech 1995] O2 Technology, "*Version Management Reference Manual*", 1995.

[Özsu, et al 1993] Özsu M.T., Peters R.J., Irani B., Lipka A., Munoz A., Szafron D., "*TIGUKAT Object Management System: Initial Design and Current Directions*", Proceedings of CASCON'93, Toronto (Canada), October 1993.

[Özsu, et al 1995] Özsu M.T., Peters R.J., Szafron D., Irani B., Lipka A., Munoz A., "*TIGUKAT: A uniform Behavioral Objectbase Management System*", The VLDB Journal, vol 4, pp 100-147, August 1995.

P

[Palisser 1989] Palisser C., "*Le Modèle de Versions du Système Charly*", Actes des 6èmes Journées Bases de Données Avancées - BDA'89, Montpellier (France), 1989.

[Papakonstantinou, et al 1995] Papakonstantinou Y., Garcia-Molina H., Widom J., "*Object Exchange Across Heterogeneous Information Sources*", IEEE International Conference on Data Engineering, pp. 251-260, Taipei (Taiwan), March 1995.

[Pedersen, Jensen 1998] Pedersen T.B., Jensen C.S., "*Research Issues in Clinical Data Warehousing*", SSDBM'98, July 1998, Capri (Italy).

[Pedersen, Jensen 1999] Pedersen T.B., Jensen C.S, "*Multidimensional Data Modeling for Complex Data*", In Proceedings of the International Conference on Data Engineering - ICDE'99, March 1999.

Q

[Quass, et al 1996] Quass D., Gupta A., Mumick I., Widom J., "*Making Views Self-Maintainable for Data Warehousing*", In Proceedings of the Conference on Parallel and Distributed Information Systems, Miami Beach (Florida, USA), December 1996.

[Quass, Widom 1997] Quass D., Widom J., "*On-Line Warehouse View Maintenance for Batch Updates*", In Proceedings of the ACM SIGMOD Conference, Tuscon (Arizona, USA), May 1997.

R

[Reanimatic 1999] Projet REANIMATIC, Action Concertée Incitative "*Télémédecine et Technologies pour la Santé*", 1999.

[Ravat 1996] Ravat F., "*OD^3 : contribution méthodologique à la conception de bases de données orientées objet réparties*", Thèse de l'Université Paul Sabatier - Toulouse III, Septembre 1996.

[Ravat, Teste 2000a] Franck Ravat, Olivier Teste, "*Object-Oriented Decision Support System*", Proceedings of the 2nd International Conference on Enterprise Information Systems - ICEIS'00, July 4-7 2000, Stafford (UK).

[Ravat, Teste 2000b] Franck Ravat, Olivier Teste, "*An Object Data Warehousing Approach: a Web Site Repository*", Proceedings of Challenges of the Enlarged 4th East-European Conference on Advances in Databases and Information Systems - ADBIS-DASFAA, September 5-8 2000, Prague (Czech Republic).

[Ravat, Teste 2000c] Franck Ravat, Olivier Teste, "*A Temporal Object-Oriented Data Warehouse Model*", Proceedings of the 11th International Conference on Database and Expert Systems - DEXA 2000, September 4-8 2000, Greenwich (London, UK).

[Ravat, Teste, Zurfluh 1999] Franck Ravat, Olivier Teste, Gilles Zurfluh, "*Towards Data Warehouse Design*", Proceedings of the 8th International Conference on Information and Knowledge Management - CIKM'99, ACM Press - *ed*. Susan Gauch - p359-366, November 2-6 1999, Kansas City (Missouri, USA).

[Ravat, Teste, Zurfluh 2000a] Franck Ravat, Olivier Teste, Gilles Zurfluh, "*Modélisation et extraction de données pour un entrepôt objet*", Actes des 16ième Journées Bases de Données Avancées - BDA'2000, 24-27 Octobre 2000, Blois (Loir et Cher, France).

[Ravat, Teste, Zurfluh 2000b] Franck Ravat, Olivier Teste, Gilles Zurfluh, " *A Temporal Object-Oriented Data Warehouse Model: Comprehensive Definition*", Rapport Interne IRIT/00-14-R, Juin 2000, Toulouse (France).

[Ravat, Teste, Zurfluh 2001] Franck Ravat, Olivier Teste, Gilles Zurfluh, "*Modélisation multidimensionnelle des systèmes décisionnels*", Accepté et à paraître dans les Actes des 1ères Journées Francophones d'Extraction et de Gestion des Connaissances - EGC 2001, 18-19 Janvier 2001, Nantes (Loire-Atlantique, France).

[Ravat, Zurfluh 1995] Ravat F., Zurfluh G. "*Répartition d'un schéma conceptuel de bases de données orientées objet*", BDA'95, pp. 145-163, Nancy (France), 1995.

[RedBrick 1998] Red Brick Systems, INC., "*Decision-Makers, Business Data and RISQL*", http://www.redbrick.com., 1998.

[Rose, Segev 1991] Rose E., Segev A., "*TOODM - A Temporal Object-Oriented Data Model with Temporal Constraints*", Proceedings of the International Conference on the Entity Relationship Approach, 1991.

[Rose, Segev 1993a] Rose E., Segev A., "*TOOA: A Temporal Object-Oriented Algebra*", Proceedings of the European Conference on Object-Oriented Programming - ECOOP'93, July 1993.

[Rose, Segev 1993b] Rose E., Segev A., "*TOOSQL - A Temporal Object-Oriented Query Language*", Proceedings of the International Conference on the Entity Relationship Approach, Dallas (Texas, USA), 1993.

[Rundensteiner, et al 1996] Rundensteiner E.A., Kuno H.A., Ra Y-G., Crestana-Taube V., Jones M.C., Marron P. J., "*The MultiView Project: Object-Oriented View Technology and Applications*", SIGMOD Conference, 1996.

S

[Samos, et al 1998] Samos J., Saltor F., Sistrac J., Bardés A., "*Database Architecture for Data Warehousing: An evolutionary Approach*", DEXA'98, Vienna (Austria), 1998.

[Schek, Pistor 1982] Schek H-J., Pistor P., "*Data Structures for an Integrated Data Base Management and Information Retrieval System*", Proceedings of the 8th International Conference on Very Large Data Bases - VLDB'82, Mexico City (Mexico), September 8-10, 1982.

[Shaw, Zdonik 1990] Shaw G.M., Zdonik, S.B., "*A Query Algebra for Object-Oriented Databases*", Proceedings of the Sixth International Conference on Data Engineering - ICDE'90, IEEE Computer Society, ISBN 0-8186-2025-0, Los Angeles (California, USA), February 5-9 1990.

[Shukla, et al 1998] Shukla A., Deshpande P.M., Naughton J.F., Ramasamy K., *"Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies*", Proceedings of VLDB'96.

[Schwer, et al. 1995] S. Schwer, N. Kraiem, J. Brunet, "*The Temporal Dimension Of Conceptual Dimension Object*" SEKE'95, 1995.

[Skarra, Zdonik 1986] Skarra A.H., Zdonik S.B., "*The management of changing types in object-oriented databases*", OOPSLA'86, pp 483-495, 1986.

[Snodgrass 1995] Snodgrass R.T. (editor), Ahn I., Ariav G., Batory D., Clifford J., Dyreson C.E., Elmasri R., Grandi F., Jensen C.S., Käfer W., Kline N., Kulkarni K., Leung T.Y.C., Lorentzos N., Roddick J.F., Segev A., Soo M.D., Sripada S.M., "*The TSQL2 Temporal Query Language*", Kluwer Academic Publishers, 1995.

[Snodgrass, et al 1998] Snodgrass R.T., Böhlen M. Jensen C. Steiner A., "*Transitioning temporal support in TSQL2 to SQL3*", in "*Temporal Databases: Research and Practice*", Springer Verlag, LNCS 1399, 1998.

[Souza 1995] Souza dos Santos C., "*Design and Implementation of Object-Oriented Views*", Proccedings of the 6th International Conference on Database and Expert Systems Applications - DEXA'95, pp 91-102, 1995.

[Souza, et al 1994] Souza dos Santos C., Abiteboul S., Delobel C, "*Virtual Schemas and Bases*", Proceedings of the 4th International Conference on Extending Database Technology - EDBT'94, 1994.

[Su, Chen 1991] Su S., Chen H., "*A Temporal Knowledge Representation Model OSAM*/T and its query language OQL/T*", Proceedings of the 17th International Conference on Very Large Databases - VLDB'91, Barcelona (Spain), September 1991.

T

[Talens, et al 1993] Talens G., Oussalah C., Colinas M.F., "*Versions of simple and composite objects*", Proceedings of the 19th International Conference on Very Large Databases - VLDB'93, Dublin (Ireland), September 1993.

[Teste 1996] Olivier Teste, "*Manipulation d'objets complexes avec le langage graphique VOHQL*", Rapport de DEA, IRIT/96-30-R, Juin 1996, Toulouse (France).

[Teste 2000] Olivier Teste, "*Elaboration d'entrepôts de données complexes*", Actes du XVIIIème Congrès INFormatique des ORganisations et Systèmes d'Information et de Décision - INFORSID'00, *ed*. INFORSID - ISBN 2-906855-16-2, p229-245, 16-19 mai 2000, Lyon (Rhône, France).

[Teste 1999a] Olivier Teste, "*Gestion du Temps dans un SGBD Relationnel : l'exemple d'Oracle*", Rapport Interne IRIT/99-06-R, Mars 1999, Toulouse (France).

[Teste 1999b] Olivier Teste, "*Modélisation Conceptuelle des Entrepôts de Données*", Rapport Interne IRIT/99-01-R, Janvier 1999, Toulouse (France).

[Theodoratos, Sellis 1997] Theodoratos D., Sellis T., "*Data warehouse configuration*", Proc. VLDB '97.

[Theodoratos, Sellis 1999] Theodoratos D., Sellis T., "*Designing Data Warehouses*", DKE 31, (3): 279-301, 1999.

[Theodoulidis, et al 1994] Theodoulidis B., Ait-Braham A., Karvelis G., "*The ORES Temporal DBMS and the ERT-SQL Query Language*", Proccedings of the 5th International Conference on Database and Expert Systems Applications - DEXA'94, Athens (Greece), 1994.

[Toobis 1998a] TOOBIS Project Deliverable, "*TODM Manual*", Work Package 3 - Task 3.4, 20 March 1998.

[Toobis 1998b] TOOBIS Project Deliverable, "*TODL Manual*", Work Package 3 - Task 3.5, 20 March 1998.

[Toobis 1998c] TOOBIS Project Deliverable, "*TOQL Manual*", Work Package 3 - Task 3.6, 20 March 1998.

[Trujillo, Palomar 1998] Trujillo J., Palomar M., "*An Object Oriented Approach to Multidimensional Database Conceptual Modeling (OOMD)*", DOLAP'98, Bethesda (Maryland, USA), November 1998.

[Tryfona, et al 1999] Tryfona N., Busborg F., Borch Christiansen J.G., "*starER: A Conceptual Model for Data Warehouse Design*", DOLAP'99, Kansas city (Missouri, USA), November 1999.

[Tsotras, Kumar 1996] Tsotras V.J., Kumar A., "*Temporal database bibliography update*", ACM SIGMOD Record, 25(1):41-51, March 1996.

U-V

[Vavouras, et al 1999] Vavouras A., Gatziu S., Dittrich K.R., "*The SIRIUS Approach for Refreshing Data Warehouses Incrementally*", BTW'99, pp 80-96, 1999.

W

[Wang, et al 1997] Wang X.S., Bettini C., Brodsky A., Jajodia S., "*Logical design for temporal databases with multiple granularities*", In Transactions on Database Systems - ACM TODS, 22(2), June 1997.

[Widom 1995] Widom J., "*Research problems in data warehousing*", Proceedings of the 4th International Conference on Information and Knowledge Management - ACM CIKM'95, November 29-December 2 1995, Baltimore (Maryland, USA).

[Widom 1996] Widom J., "*The Starburst Active Database Rule System*", IEEE Transactions on Knowledge and Data Engineering, 8(4):583-595, August 1996.

[Wiener, et al 1996] Wiener J. L., Gupta H., Labio W. J., Zhuge Y., Garcia-Molina H., Widom J., "*A System Prototype for Warehouse View Maintenance*", In Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications, pp. 26-33, Montreal (Canada), June 7 1996.

[Wu, Buchmann 1997] Wu M-C., Buchmann A.P., "*Reseach Issues in Data Warehousing*", BTW'97, pp 61-87, 1997.

[Wuu, Dayal 1993] Wuu G., Dayal U., "*A Uniform Model for Temporal and Versioned Object-oriented Databases*", Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings, pp 230-247, 1993.

X-Y

[Yang, et al 1997] Yang J., Karlapalem K., Li Q., "*Algorithms for materialized view design in data warehousing environment*", Proc. VLDB '97.

[Yang, Widom 1998] Yang J., Widom J., "*Maintaining Temporal Views Over Non-Temporal Information Sources For Data Warehousing*", In Proceedings of the 6th International Conference on Extending Database Technology, Valencia, Spain, March 1998.

[Yang, Widom 2000] Yang J., Widom J., "*Temporal View Self-Maintenance in a Warehousing Environment*", In Proceedings of the 7th International Conference on Extending Database Technology - EDBT 2000, Konstanz (Germany), March 2000.

Z

[Zhou, et al 1995a] Zhou G., Hull R., King R., Franchitti J.C., "*Using object matching and materialization to integrate heterogeneous databases*", Proceedings of the 3rd International Conference on Cooperative Information Systems - CoopIS'95, pages 4-18, 1995.

[Zhou, et al 1995b] Zhou G., Hull R., King R., Franchitti J.C., "*Data integration and warehousing using H2O*", IEEE Bulletin of the Technical Committee on Data Engineering, 18(2), 29-40, 1995.

[Zhou, et al 1996] Zhou G., Hull R., King R., "*Generating Data Integration Mediators that Use Materialization*", Journal of Intelligent Information Systems, Volume 6(2), pages 199-221, May 1996.

[Zhuge, et al 1995] Zhuge Y., Garcia-Molina H., Hammer J., Widom J., "*View Maintenance in a Warehousing Environment*", In Proceedings of the ACM SIGMOD Conference, San Jose (California, USA), May 1995.

[Zhuge, et al 1996] Zhuge Y., Garcia-Molina H., Wiener J. L., "*The Strobe Algorithms for Multi-Source Warehouse Consistency*", In Proceedings of the Conference on Parallel and Distributed Information Systems, Miami Beach (Florida, USA), December 1996.

[Zhuge, et al 1997] Zhuge Y., Wiener J. L., Garcia-Molina H., "*Multiple View Consistency for Data Warehousing*", In Proceedings of the International Conference on Data Engineering, Binghamton (UK), April 1997.

[Zhuge, et al 1998] Zhuge Y., Garcia-Molina H., Wiener J. L., "*Consistency Algorithms for Multi-Source Warehouse View Maintenance*", Journal of Distributed and Parallel Databases, volume 6, number 1, January 1998.

[Zhuge, Garcia-Molina 1998] Zhuge Y., Garica-Molina H., "*Graph Structured Views and Their Incremental Maintenance*", In Proceedings of the International Conference on Data Engineering, Orlando (Florida, USA), February 1998.

ANNEXE A :
EXEMPLE D'UNE SOURCE GLOBALE DE DONNEES MEDICALES

Exemple d'une source globale de données médicales

La source globale, c'est à dire la base de données à partir de laquelle est élaboré l'entrepôt, est décrite avec un modèle de données orientées objet. Nous justifions notre choix par le fait que le paradigme objet s'est révélé parfaitement adapté pour l'intégration de sources hétérogènes et permet de modéliser des sources intégrant des structures complexes.

Pour décrire les aspects statiques de cette source, nous utilisons le modèle objet standard : modèle de l'ODMG [Cattel 1995] étendu par le concept de composition d'objets [Bertino, et al 1998].

Les travaux, présentés dans ce document, s'appuient sur la source de données médicales suivante. Cette source de données décrit des patients et des praticiens. Ces derniers travaillent dans des services hospitaliers. Chaque service est dirigé par un praticien. Les patients consultent les praticiens, qui émettent des diagnostics et des prescriptions médicamenteuses.

```
interface PERSONNE {
   attribute String nom;
   attribute String prenom;
   attribute Struct T_Adresse {
      String libelle,
      Short code,
      String ville } adresse;
   attribute Struct T_Date {
      Short annee,
      Short mois,
      Short jour } naissance;
   // opérations
   Short age();
   Boolean est_adulte();
   }

interface PATIENT extends PERSONNE {
   attribute Short num_secu;
   attribute Short cle_secu;
   composite dependent exclusive {hierarchy(PATIENT)}
             relationship CARNET_SANTE carnet
             inverse CARNET_SANTE::proprietaire;
   relationship Set<VISITE> consultations
                inverse VISITE::patient;
   // opérations
   Short nb_visites();
   }

interface CARNET_SANTE {
```

```
   attribute Struct T_Vaccin {
      String type_vaccin,
      Struct T_Date {
         Short annee,
         Short mois,
         Short jour } date } vaccination;
   attribute List<String> allergie;
   relationship PATIENT proprietaire
                inverse PATIENT::carnet;
   // opérations
   Struct T_Date{Short annee,Short mois,Short jour}
                             prochaine_vaccination(String type_vac);
   Boolean vaccination_a_jour(String type_vac);
   }

interface PRATICIEN extends PERSONNE {
   attribute Short num_prat;
   attribute String categorie;
   attribute String specialite;
   attribute Short type_convention;
   attribute List<String> diplomes;
   relationship CABINET travaille
                inverse CABINET::praticiens;
   relationship SERVICE intervient
                inverse SERVICE::equipe;
   relationship SERVICE dirige
                inverse SERVICE::directeur;
   relationship Set<VISITE> consultations
                inverse VISITE::praticien;
   // opérations
   Boolean est_conventionne();
   Short nb_services_diriges();
   String la_specialite();
   Double taux_teletransmission();
   Double ratio_teletramission()
   Double ratio_generique();
   }

interface CABINET {
   attribute String nom;
   attribute Struct T_Adresse {
      String libelle,
      Short code,
      String ville } adresse;
   relationship Set<PRATICIEN> praticiens
                inverse PRATICIEN::travaille;
```

```
   // opérations
   Boolean est_rural();
   Boolean est_urbain();
   Short nb_praticiens();
   List<String> specialites();
   }

interface ETABLISSEMENT {
   attribute String nom;
   attribute Struct T_Adresse {
      String libelle,
      Short code,
      String ville } adresse;
   attribute String statut;
   attribute Struct T_subvention {
      Double montant,
      String institution,
      String type} subvention;
   composite dependent exclusive {hierarchy(ETABLISSEMENT)}
             relationship Set<SERVICE> organisation
             inverse SERVICE::etablissmt
             delete restrict;
   // opérations
   Short nbServices();
   Double budget();
   Boolean estPublic();
   Double taxe();
   String depart();
   String region()
   }

interface SERVICE {
   attribute String nom;
   attribute Double budget;
   relationship ETABLISSEMENT etablissmt
                inverse ETABLISSEMENT::organisation;
   relationship Set<PRATICIEN> equipe
                inverse PRATICIEN::intervient;
   relationship PRATICIEN directeur
                inverse PRATICIEN::dirige;
   // opérations
   Short tailleEquipe();
   Double budget();
   Boolean estPublic();
   }
```

```
interface VISITE {
   attribute String reference;
   attribute Double honoraire;
   attribut Boolean teletransmise;
   attribute Set<String> symptomes;
   attribute Double poids;
   attribute Double taille;
   attribute Double temperature;
   attribute Struct T_Tension {
      Short max,
      Short min } tension;
   attribute String diagnostic;
   relationship PRATICIEN praticien
                inverse PRATICIEN::consultations;
   relationship PATIENT patient
                inverse PATIENT::consultations;
   relationship Set<MEDICAMENT> prescription
                inverse MEDICAMENT::prescrit;
   // opérations
   Boolean hypertension();
   Boolean est_obese();
   Double tarif();
   Double montant_prescriptions();
   Double depassement();
   Boolean est_teletransmisse();
   }

interface MEDICAMENT {
   attribute String code;
   attribute Short categorie;
   attribute Double tarif;
   attribute Double taux_secu;
   attribute Short type_molecule;
   attribute Boolean generique;
   attribute String posologie;
   attribute Short quantite;
   attribute List<String> actions;
   attribute List<String> effets_indesirables;
   // opérations
   Double tarif();
   Boolean est_generique();
   }
```

ANNEXE B :
QUELQUES OFFRES COMMERCIALES D'OUTILS OLAP

Quelques offres commerciales d'outils OLAP

Ces dernières années, les vendeurs du marché ont développé différents produits adoptant des solutions et des architectures variées. Ces approchent portent tant sur les systèmes propriétaires MOLAP que sur les SGBDR (Systèmes de Gestion de Bases de Données Relationnelles) ou encore sur des applications qui comportent des fonctionnalités OLAP à différents niveaux. Les outils issus de l'industrie se concentrent essentiellement sur deux aspects : le stockage et l'interrogation.

1 OUTILS DE STOCKAGE

Oracle compte parmi les sociétés qui occupent les parts de marché les plus importantes (20,7% en 1997)[14] avec son offre Oracle Express Server (produit acquis de la société IRI Software en 1995). Puisque cet outil nous paraît une des offres les plus complète et représentative du marché des logiciels OLAP, nous avons présenté en détail quelques composantes essentielles d'Oracle Express Server dans la section 3.5 du chapitre 1 (présentant l'état de l'art).

Sociétés	Serveur	Architecture	Avantages
Hyperion Solution	Essbase Server	MOLAP	Politique de sécurité flexible (définition fine de permissions d'accès aux données)
Applix	OLAP TM-1	MOLAP	Taille des bases réduite en raison du concept RAP (opération OLAP se fait en temps réel) et de l'absence de pré-agrégats Stockage de vues virtuelles (plus lentes)
Oracle	Express Server, Personnal Server	MOLAP, DOLAP	Intégration d'un grand nombre de sources variées (multidimensionnelles, relationnelles, feuilles de calculs, fichiers plat…)
MicroStrategy	DSS Server	ROLAP	Accès à un grand nombre de bases relationnelles Supporte d'importants volumes de données (téraoctets)
MicroSoft	SQL Server 7 (OLAP Services 7)	ROLAP ou MOLAP	Coûts très modérés SQL Server très répandu dans les PME
SAS Institue	MDDB	MOLAP ou ROLAP	_

Tableau 32 : Offres commerciales de serveurs pour les applications OLAP.

[14] *http://www.olapreport.com/index.htm*

Néanmoins, d'autres constructeurs proposent des solutions concurrentes. Nous présentons brièvement quelques produits dans le tableau 1.

Le terme DOLAP (Desktop OLAP) désigne un petit produit OLAP faisant de l'analyse multidimensionnelle en local. Les opérations sont exécutées localement, par opposition à une configuration client/serveur. Il peut y avoir une base multidimensionnelle (façon Personal Express), ou bien de l'extraction de cube (façon Business Objects).

2 OUTILS D'INTERROGATION

Des requêteurs et outils OLAP sont proposés facilitant la construction de requêtes SQL (jointures masquées, alias de noms d'attributs,…) et les analyses multidimensionnelles en proposant des objets graphiques (tableaux croisés,…). Ces outils utilisent les données de l'entrepôt ou des magasins, mais ne permettent pas leur construction.

Face à la multitude d'acteurs sur le marché et aux offres logicielles innombrables, nous présentons trois produits qui nous semble représentatifs des différentes offres, sans prétendre être exhaustif.

Sociétés	Produits logiciels	Architecture	Caractéristiques
Cognos	Impromptu, PowerPlay	DOLAP	Requêteur visuel pour l'extraction des données Stockage des informations sur une machine locale Fonctionnalités OLAP (visualisation, manipulation…)
Business Objects	Business Objects	DOLAP	Organisation de l'information et manipulations orientées utilisateurs Fonctionnalités OLAP (visualisation, manipulation, reporting…)
Oracle	Express Analyzer, Web Agent	MOLAP, DOLAP	Manipulations multidimensionnelles des données Fonctionnalités de reporting Langage de programmation L4G Publication Web

Tableau 33 : Offres commerciales d'applications OLAP.

Résumé

Le mémoire de cette thèse traite de la modélisation conceptuelle et de la manipulation des données (par des algèbres) dans les systèmes d'aide à la décision. Notre thèse repose sur la dichotomie de deux espaces de stockage : l'**entrepôt de données** regroupe les extraits des bases sources utiles pour les décideurs et les **magasins de données** sont déduits de l'entrepôt et dédiés à un besoin d'analyse particulier.

Au niveau de l'entrepôt, nous définissons un modèle de données permettant de décrire l'évolution temporelle des objets complexes. Dans notre proposition, l'**objet entrepôt** intègre des états courants, passés et archivés modélisant les données décisionnelles et leurs évolutions. L'extension du concept d'objet engendre une extension du concept de classe. Cette extension est composée de **filtres** (temporels et d'archives) pour construire les états passés et archivés ainsi que d'une **fonction de construction** modélisant le processus d'extraction (origine source). Nous introduisons également le concept d'**environnement** qui définit des parties temporelles cohérentes de tailles adaptées aux exigences des décideurs. La manipulation des données est une extension des algèbres objet prenant en compte les caractéristiques du modèle de représentation de l'entrepôt. L'extension se situe au niveau des opérateurs temporels et des opérateurs de manipulation des ensembles d'états.

Au niveau des magasins, nous définissons un modèle de données multidimensionnelles permettant de représenter l'information en une **constellation** de **faits** ainsi que de **dimensions** munies de **hiérarchies multiples**. La manipulation des données s'appuie sur une algèbre englobant l'ensemble des opérations multidimensionnelles et offrant des opérations spécifiques à notre modèle. Nous proposons une **démarche d'élaboration** des magasins à partir de l'entrepôt.

Pour valider nos propositions, nous présentons le **logiciel GEDOOH** (Générateur d'Entrepôts de Données Orientées Objet et Historisées) d'aide à la conception et à la création des entrepôts dans le cadre de l'application médicale **REANIMATIC**.